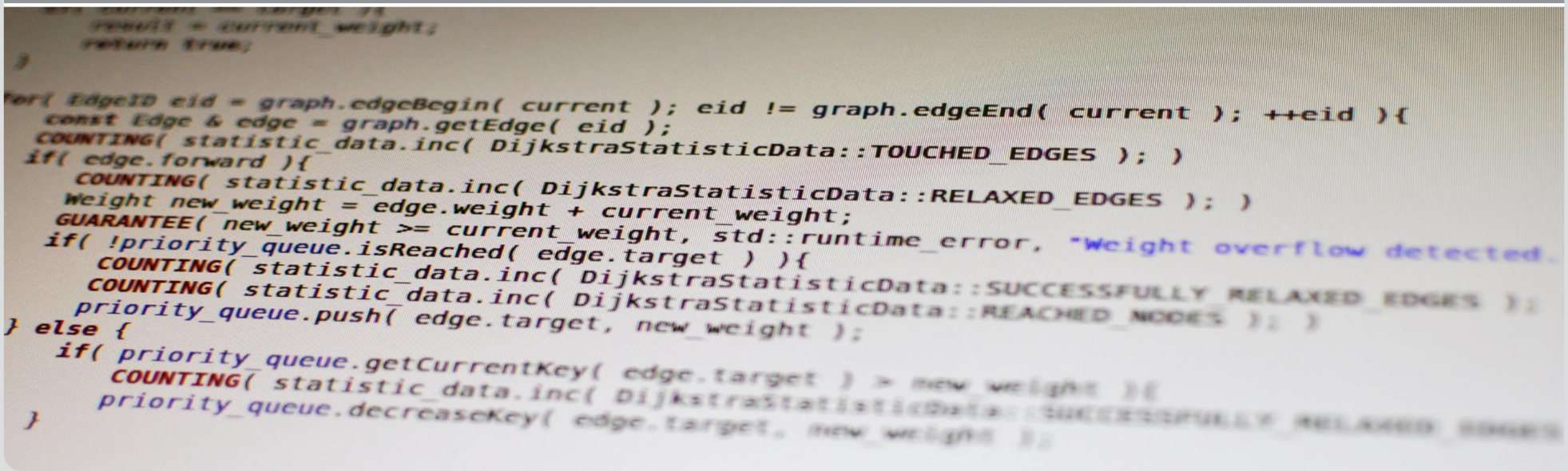
Karlsruher Institut für Technologie

Übung 4 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – *{moritz.laupichler, hans-peter.lehmann}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS22.php`

Institut für Theoretische Informatik - Algorithmik II

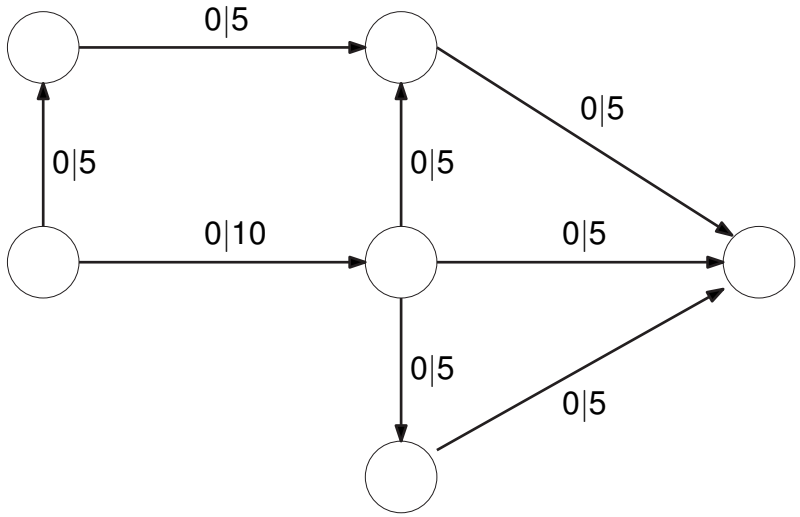


Themenübersicht

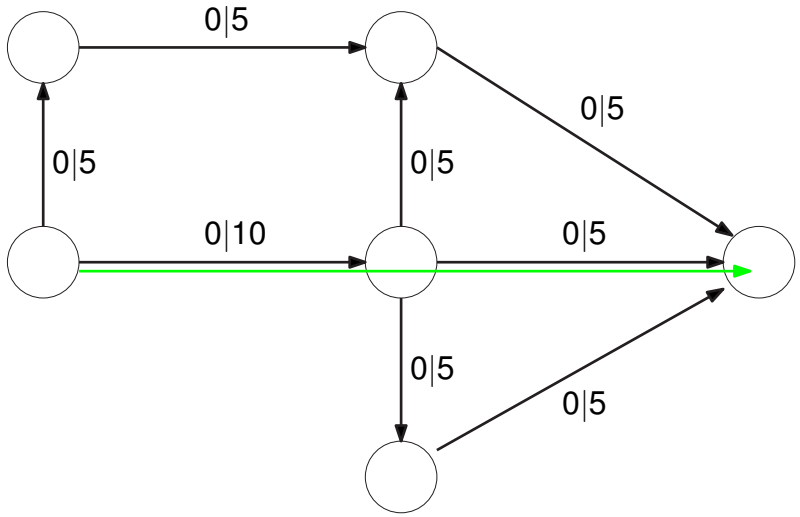
- Maximale Flüsse
  - Ford-Fulkerson
  - Dinitz
  - Preflow-Push
- Randomisierte Algorithmen
  - Las Vegas, Monte Carlo
  - Matrizen
  - Coupons
  - ...

Ford Fulkerson

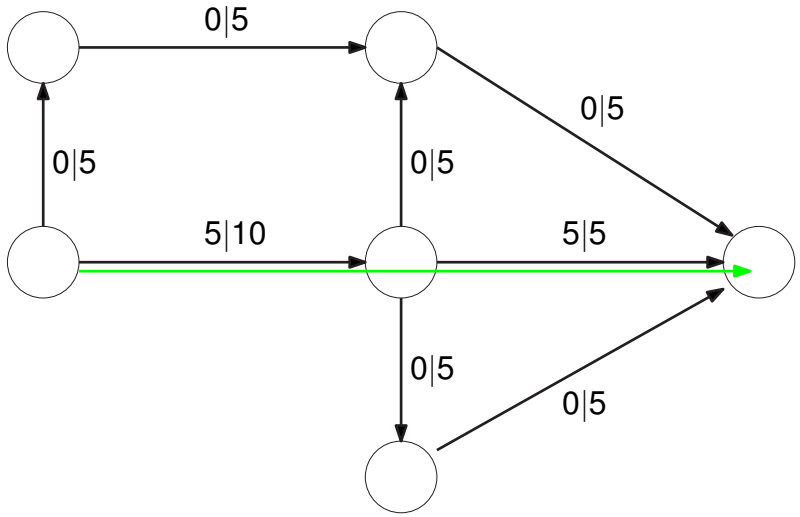
Flüsse und Ford Fulkerson



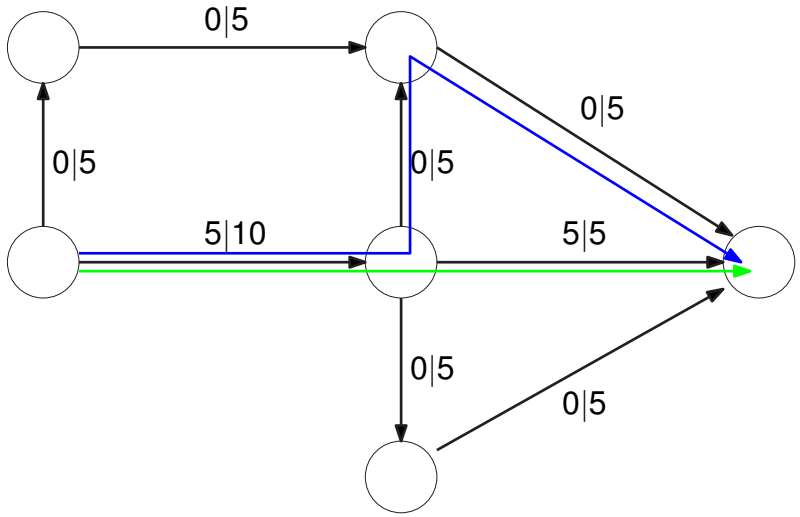
Flüsse und Ford Fulkerson



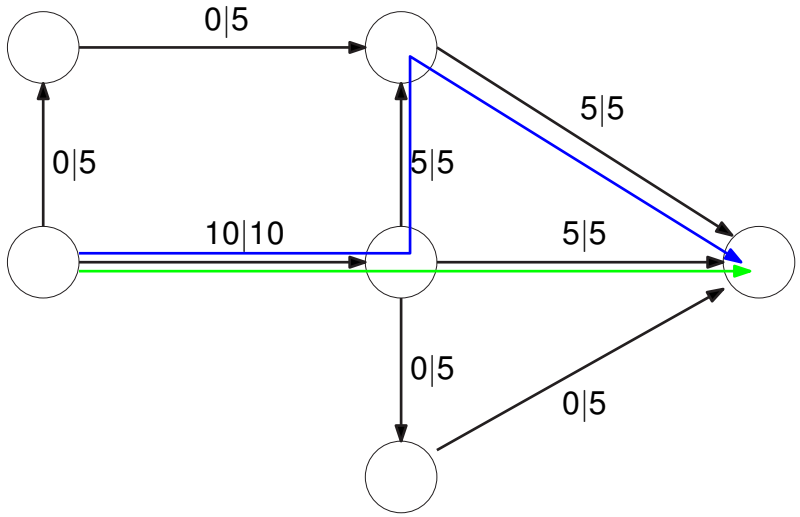
Flüsse und Ford Fulkerson



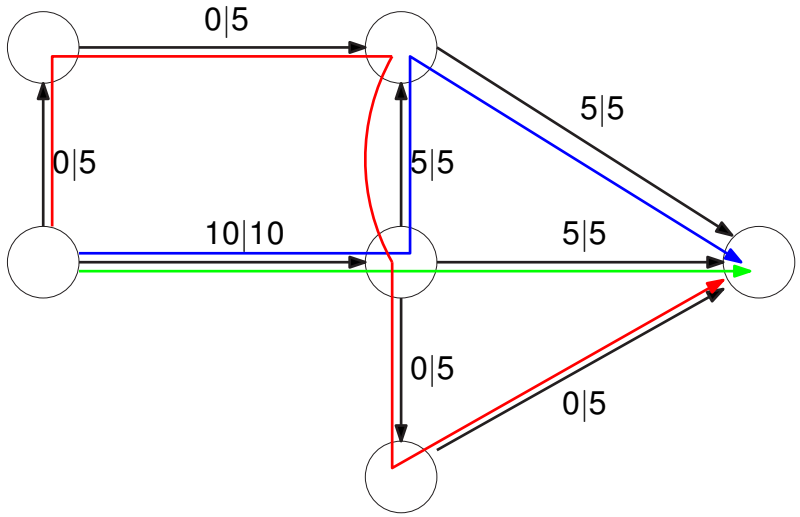
Flüsse und Ford Fulkerson



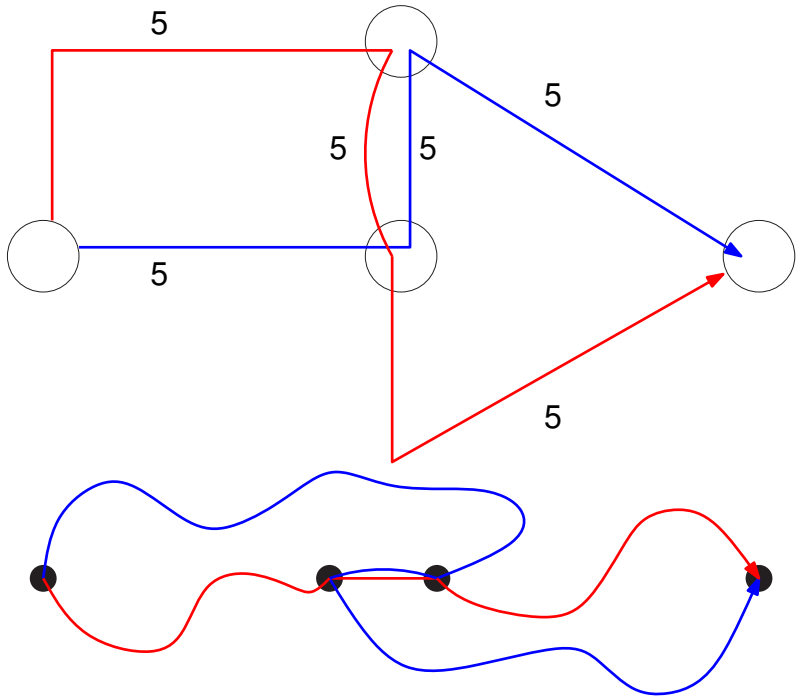
Flüsse und Ford Fulkerson



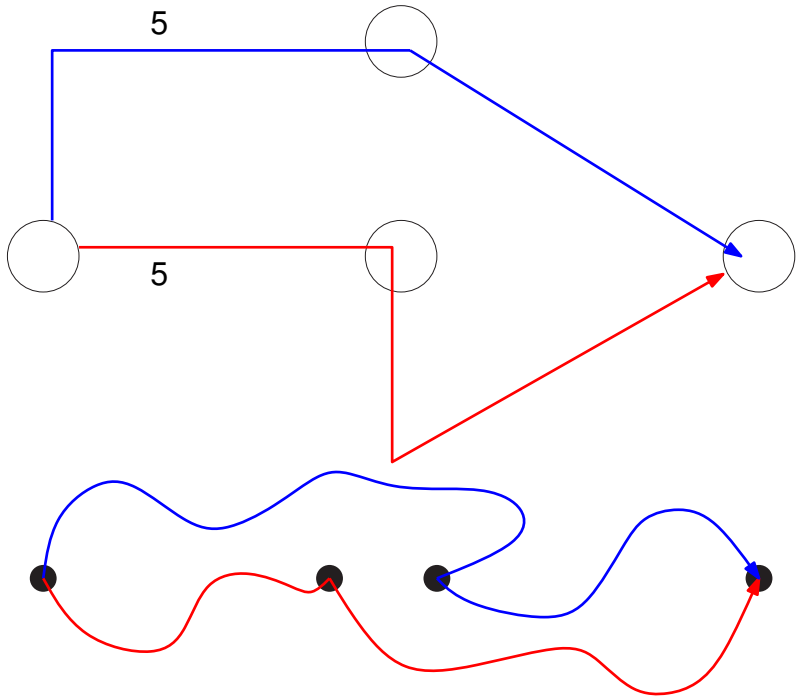
Flüsse und Ford Fulkerson



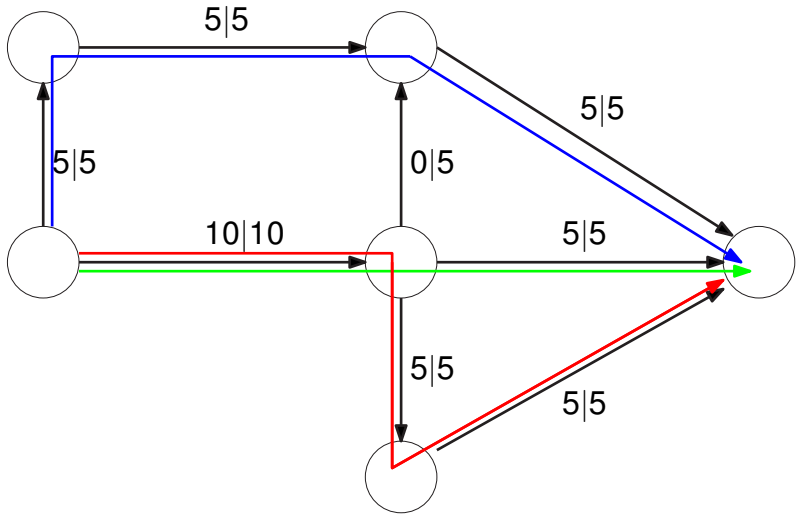
Flüsse und Ford Fulkerson



Flüsse und Ford Fulkerson



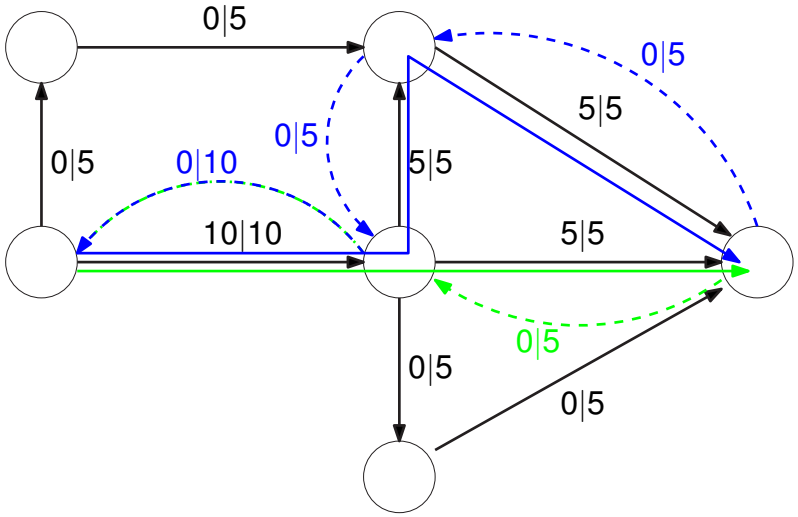
Flüsse und Ford Fulkerson



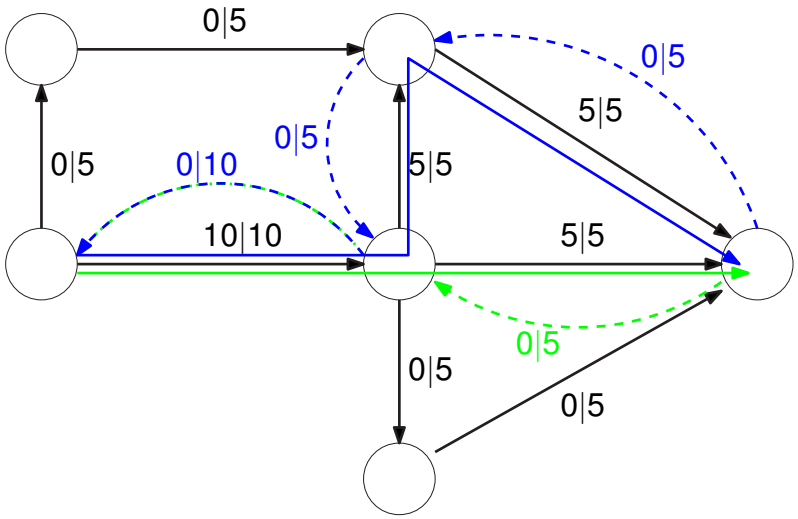
Residualgraph

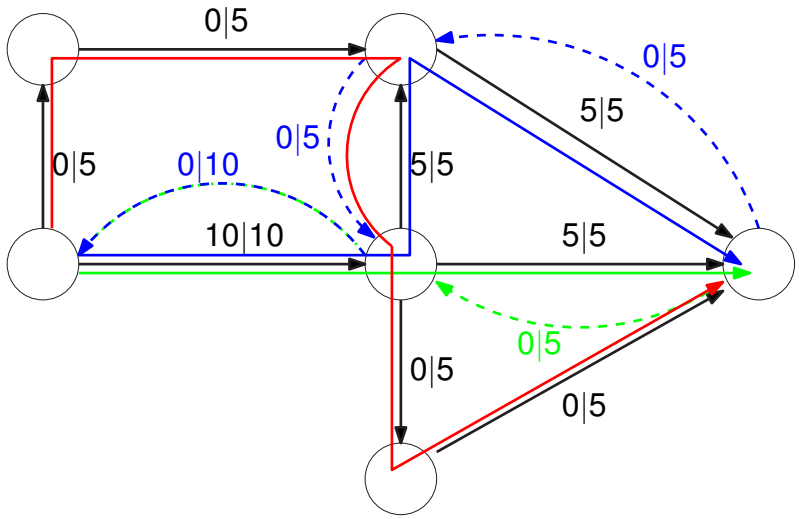
- Verwalten von Restkapazitäten
- Modellierung und Erkennung von “Gegenflüssen”
- $c^f(e) = c(e) - f(e)$: Restkapazität
  Hier: Fluss $f(e)$ und Gesamtkapazität $c(e)$
- $c^f(e^{\text{rev}}) = f(e)$: Fluss über Kante e
  Hier: Restkapazität und Gesamtkapazität von e
- Keine 0-Gewicht Kanten
- Flüsse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

Flüsse und Ford Fulkerson

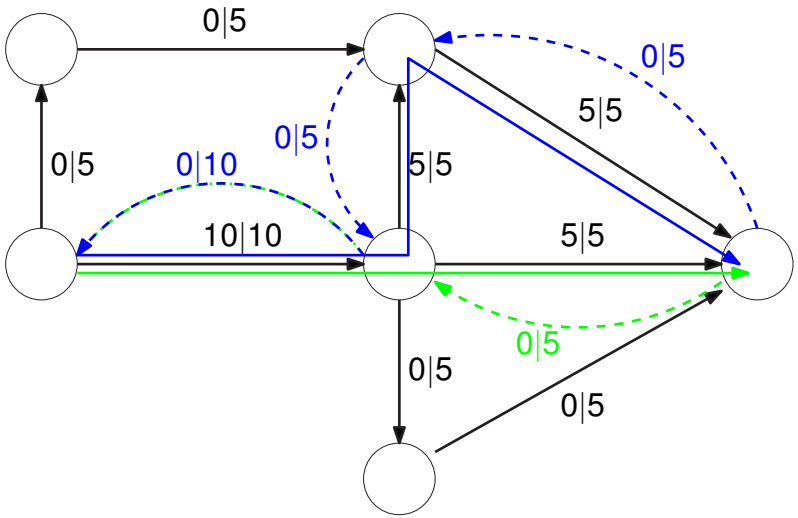


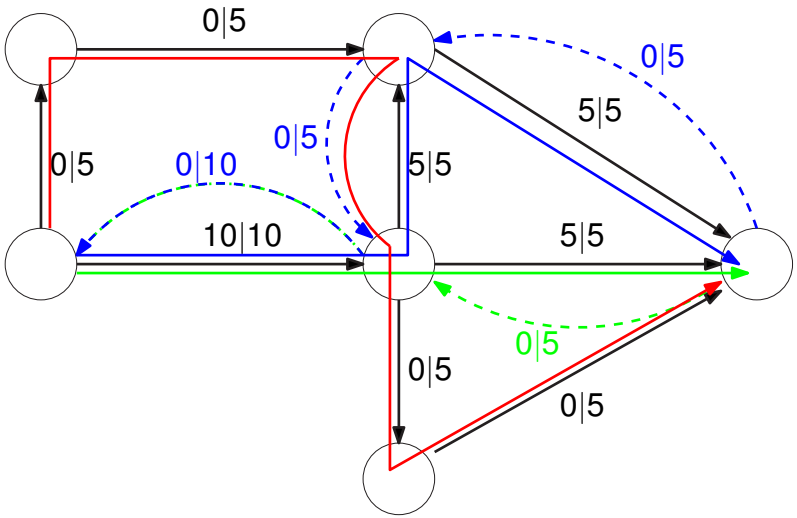
Flüsse und Ford Fulkerson

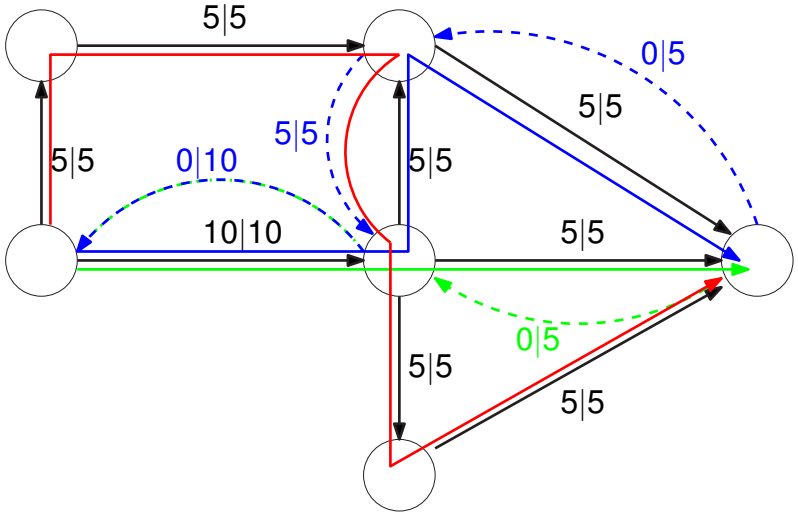




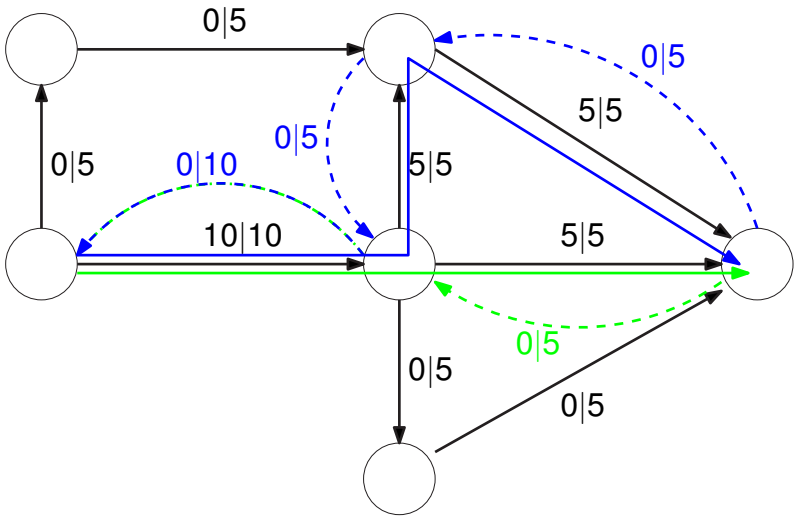
Flüsse und Ford Fulkerson

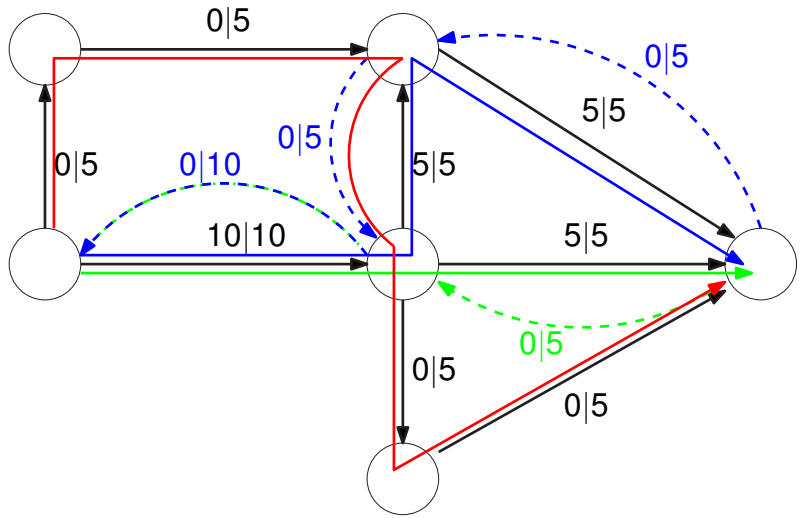


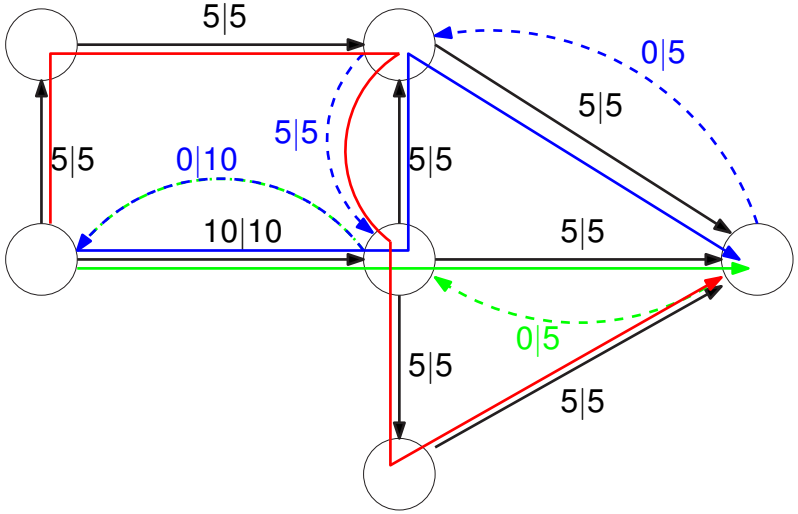


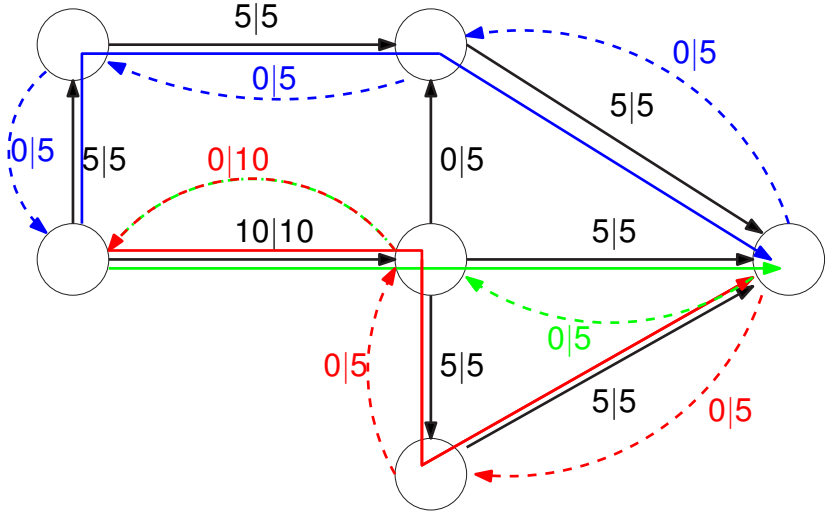


Flüsse und Ford Fulkerson



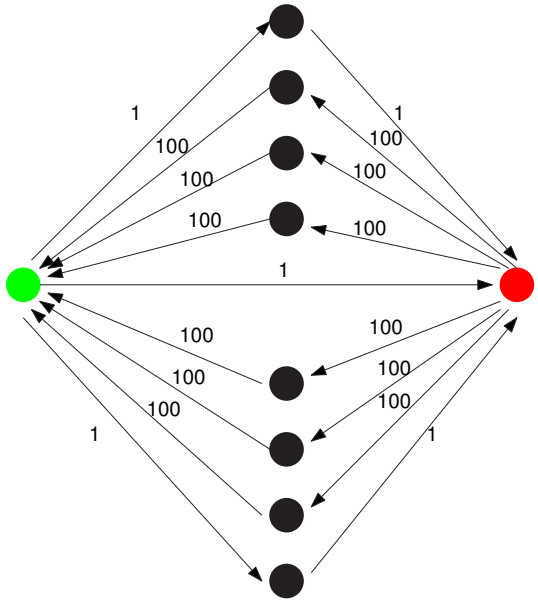






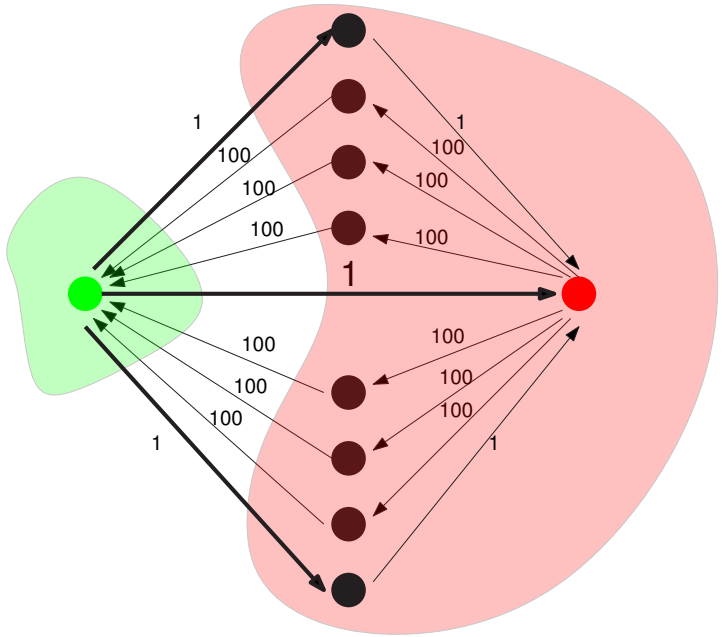
Max Flow - Min Cut

Max Flow - Min Cut



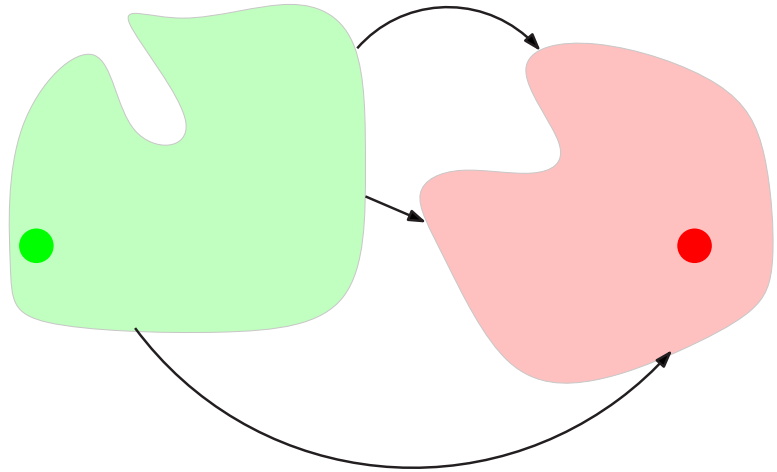
Max Flow - Min Cut

- S-T-Schnitte betrachten nur Kanten $S \to T$
- Kanten $T \to S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S-T-Schnitte getrennt
- Flow muss von s nach t, auch durch alle möglichen S-T-Schnitte
  $\to$ 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
- s und t werden durch alle möglichen S-T-Schnitte getrennt
- Flow muss von s nach t, auch durch alle möglichen S-T-Schnitte
  $\to$ Max Flow = Min S-T-Cut



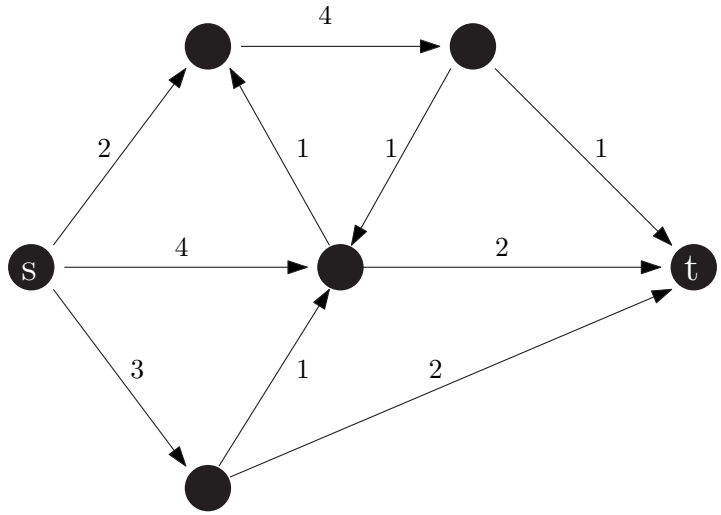
Dinitz Algorithmus

Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke *t* an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

  auch: kürzeste Wege Netzwerk, Schichtgraph

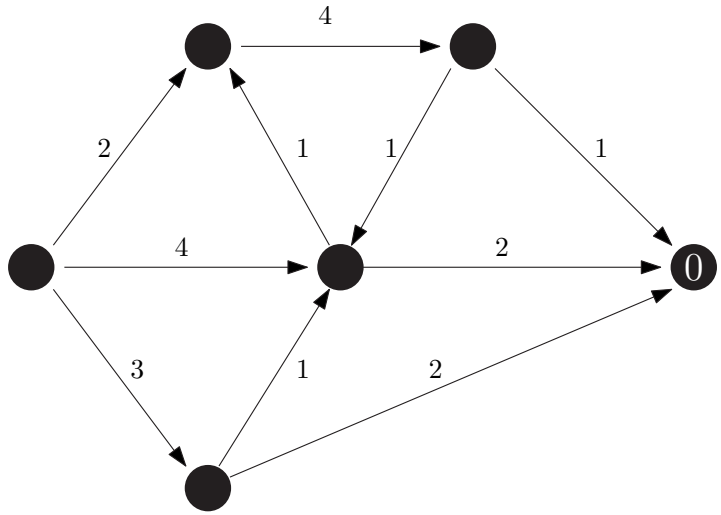


Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten t
  - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

  auch: kürzeste Wege Netzwerk, Schichtgraph

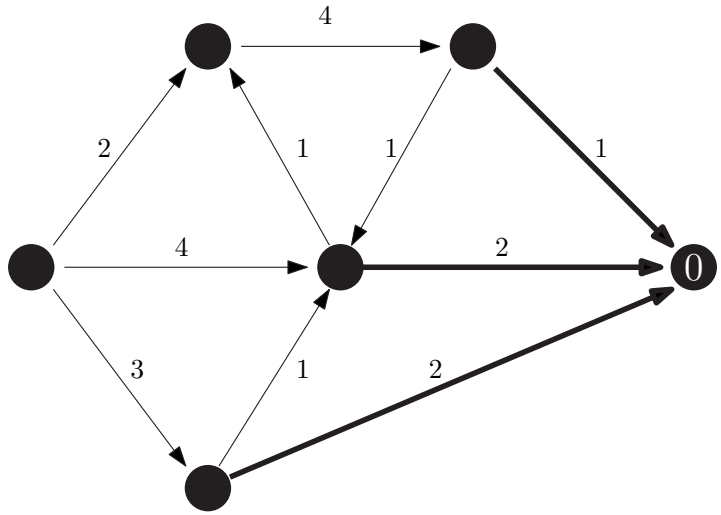


Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke *t* an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

  auch: kürzeste Wege Netzwerk, Schichtgraph

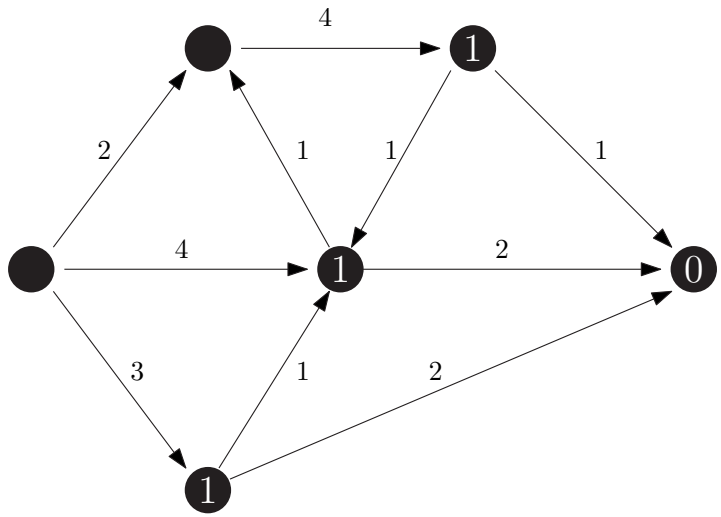


Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke *t* an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

  auch: kürzeste Wege Netzwerk, Schichtgraph

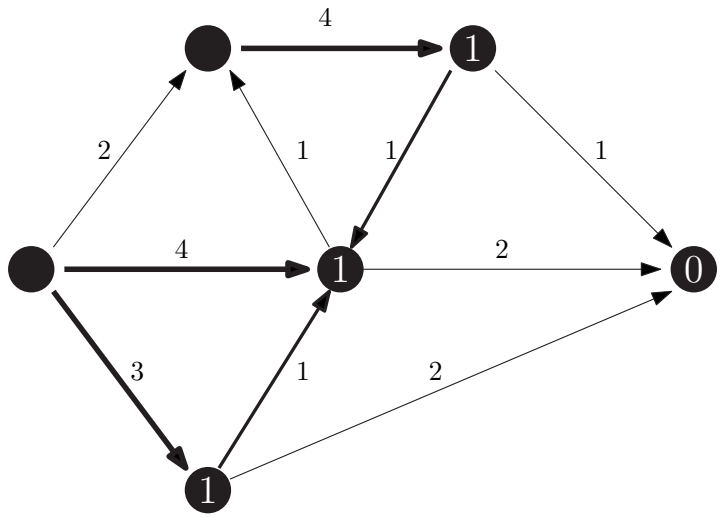


Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke *t* an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

  auch: kürzeste Wege Netzwerk, Schichtgraph

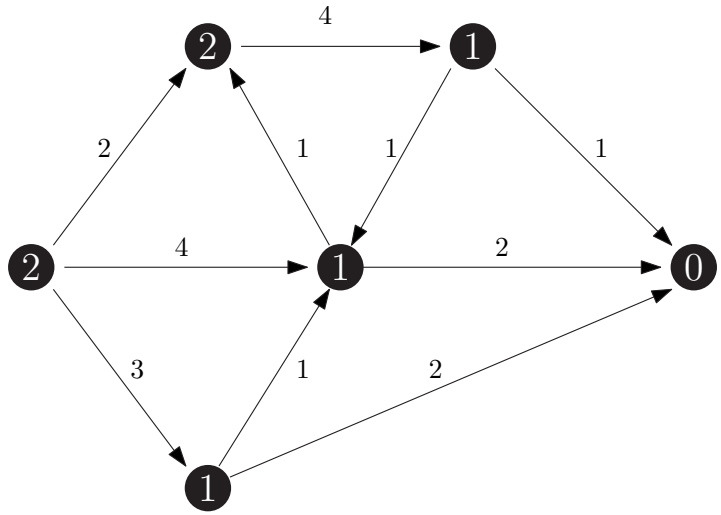


Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke *t* an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

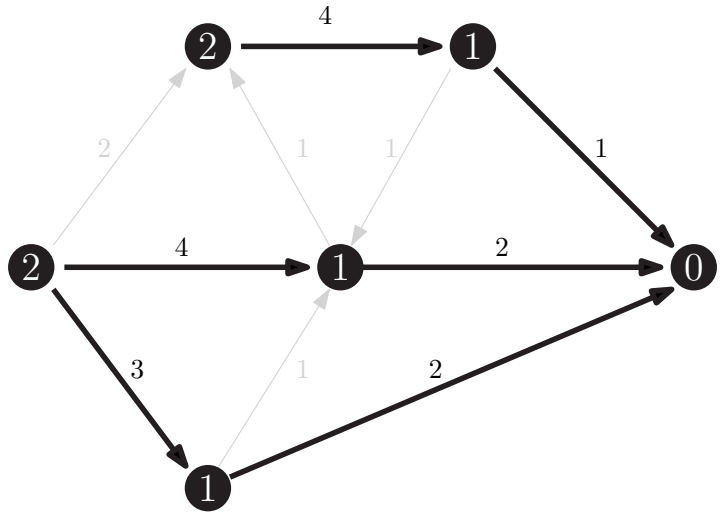
  auch: kürzeste Wege Netzwerk, Schichtgraph



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 $E_S = E' \cup E'^{rev}$
- betrachte Analogie zu Edmonds-Karp (kürzeste erhöhende Wege)

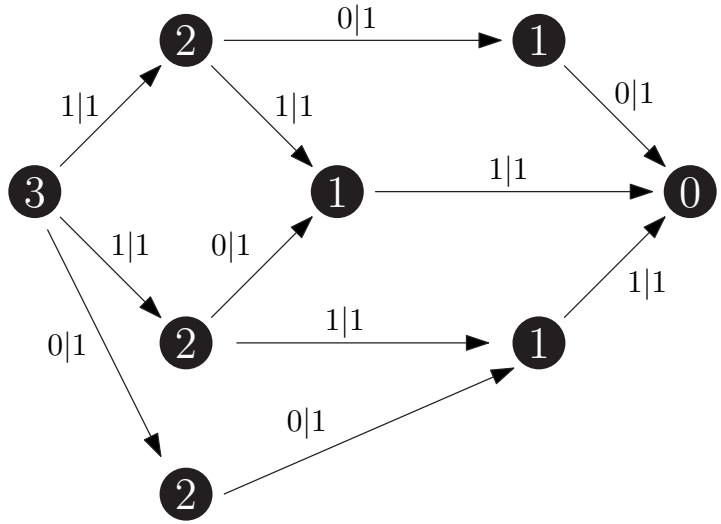


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph

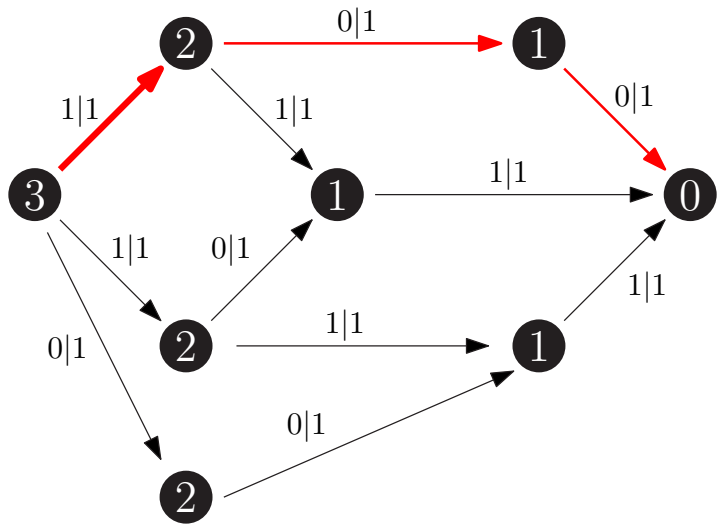


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph

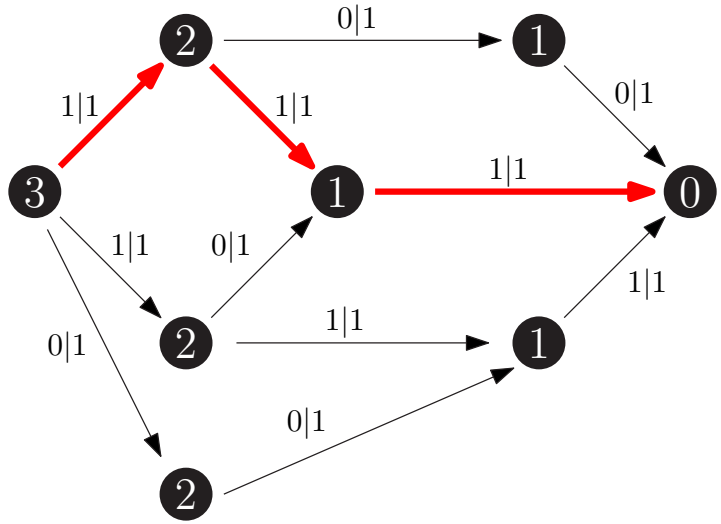


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph

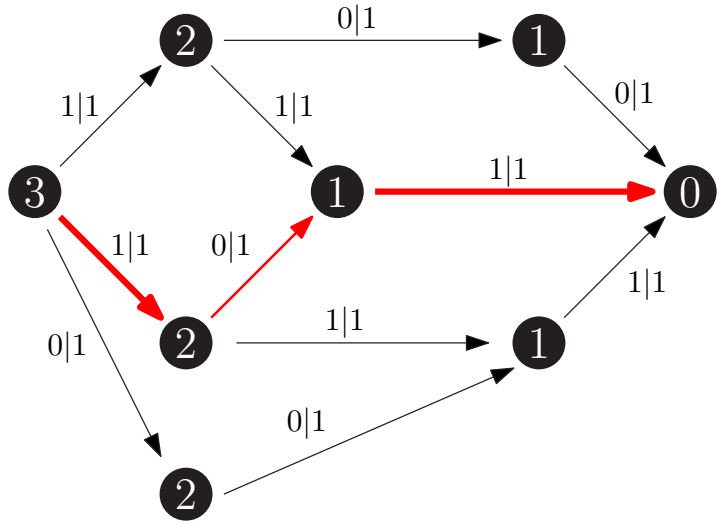


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph

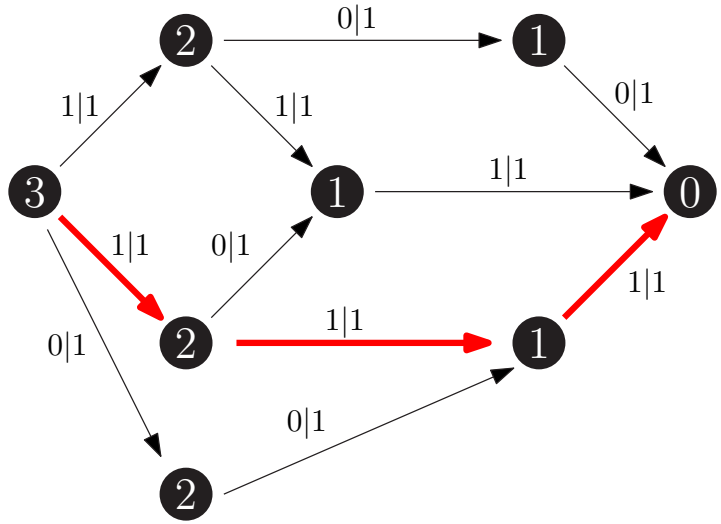


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph

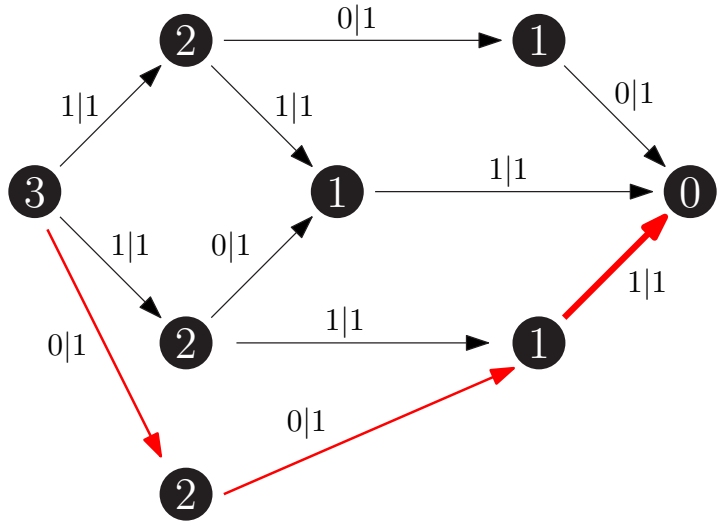


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

  “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

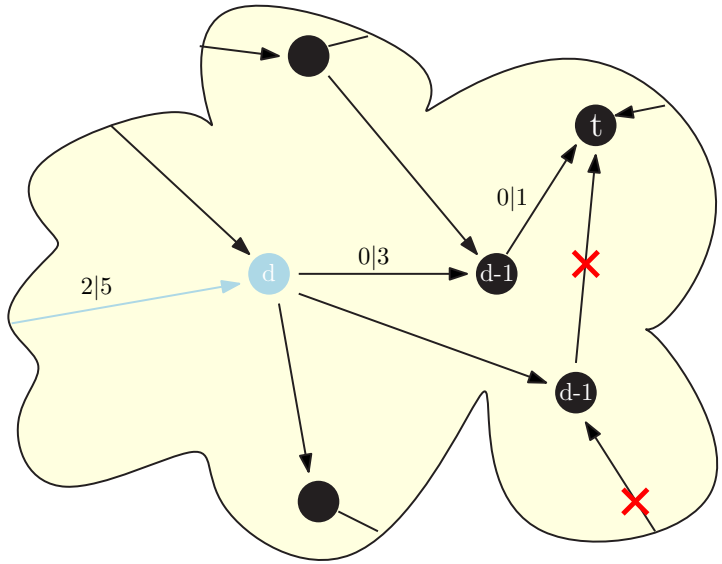
Blockierender Fluss: Operationen

- *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

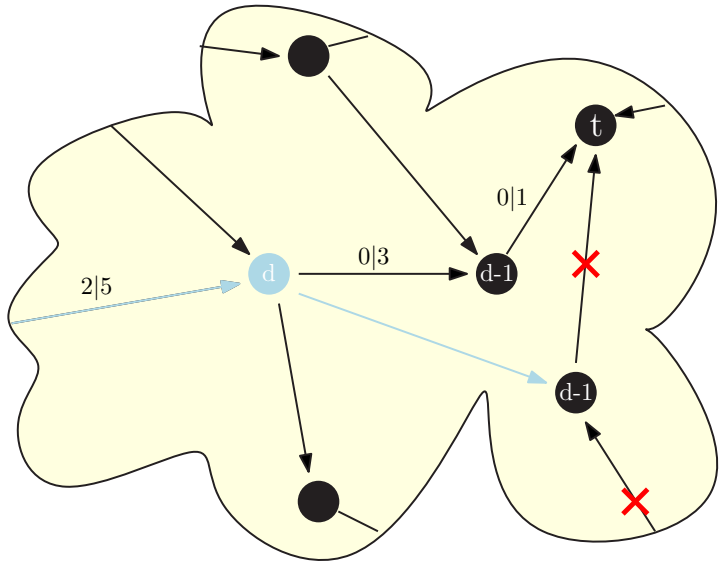
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

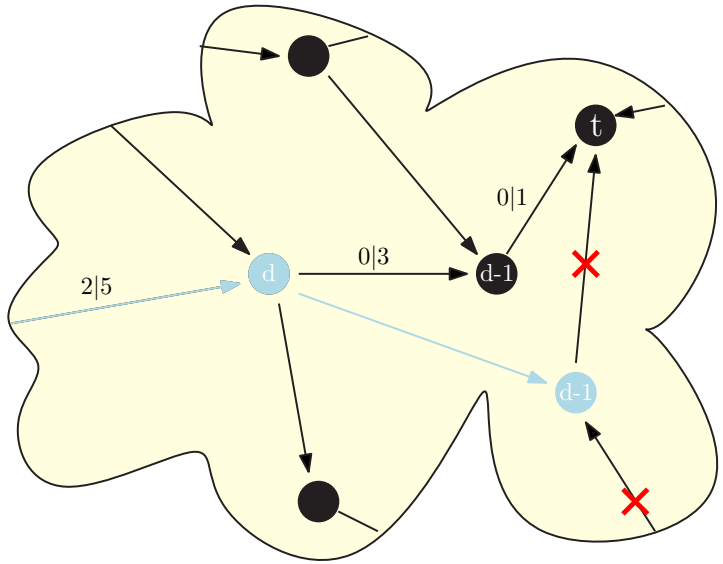
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

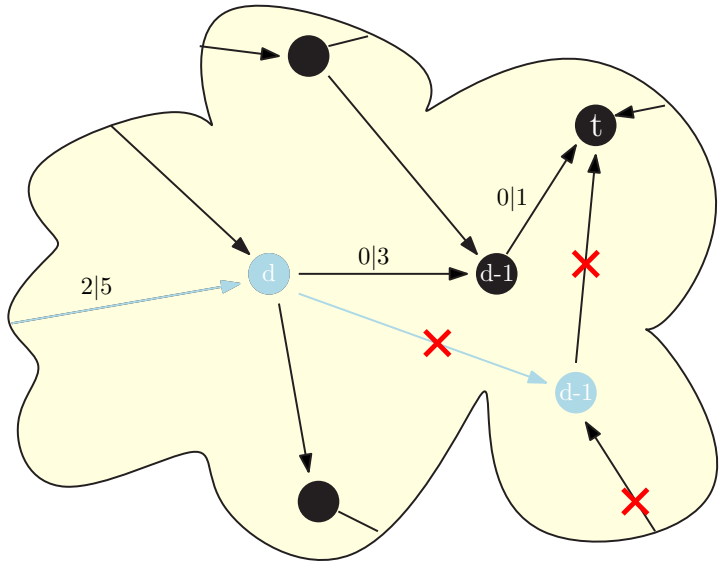
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

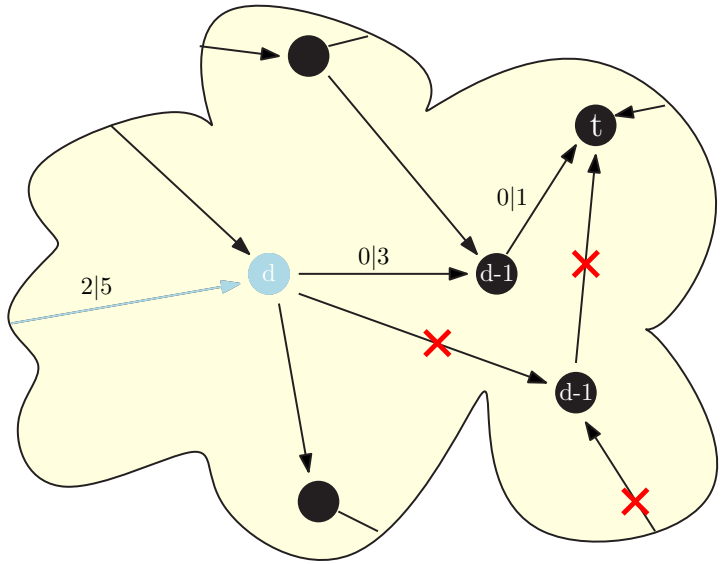
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

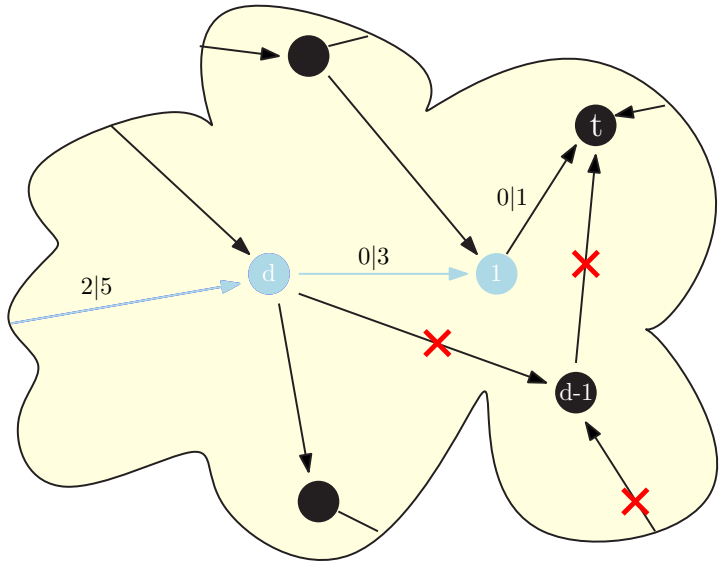
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

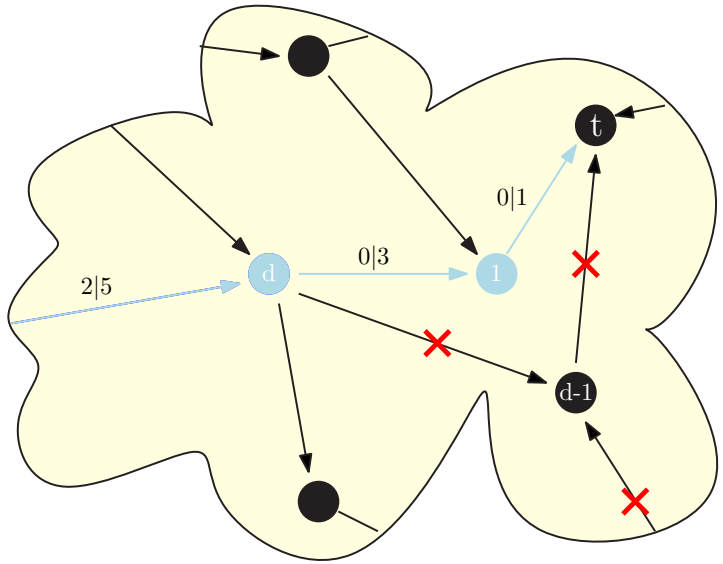
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

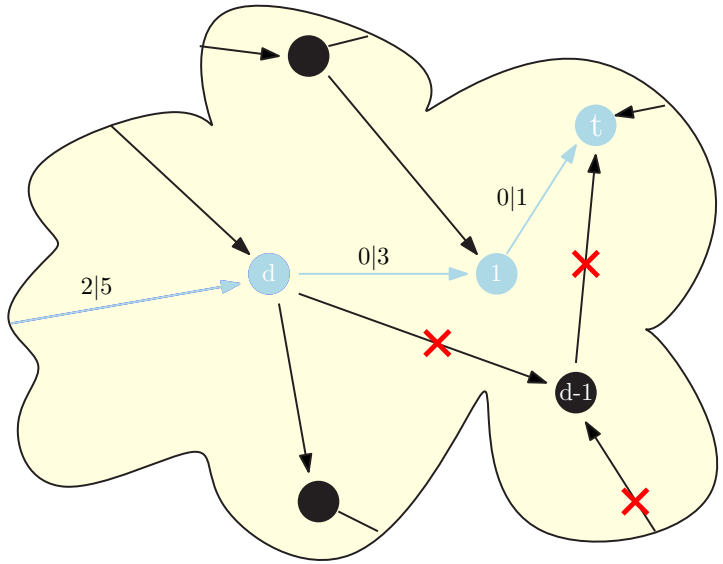
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

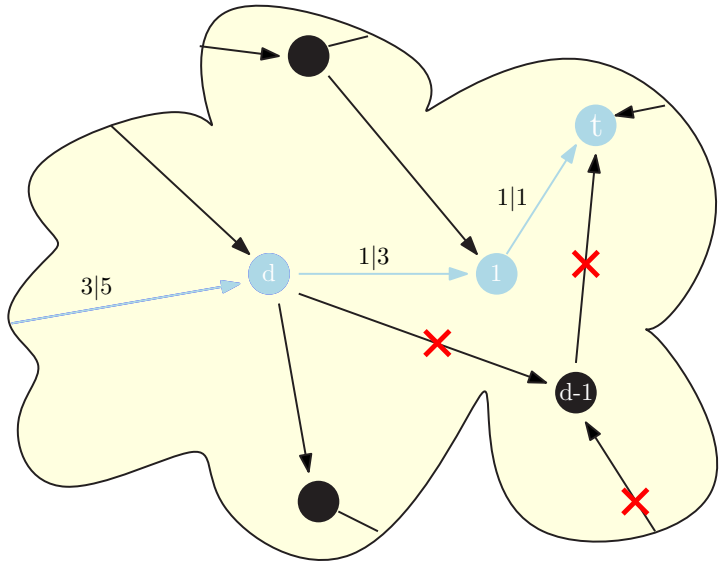
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

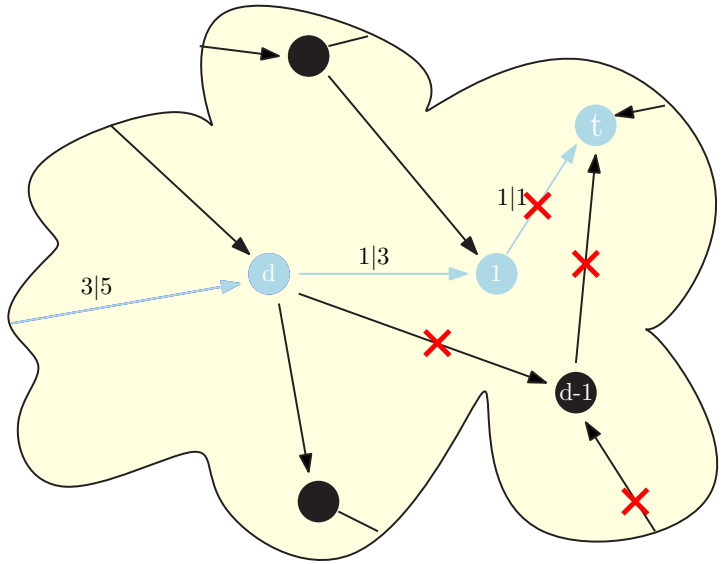
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



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**
    Ohne breakthrough nur retreats
  - **Breakthrough:** Vorher $\leq n$ **erfolgreiche extends**
    Schichtgraph schließt Kreise aus
  - **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, wie in Vorlesung, nicht hier)

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

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

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

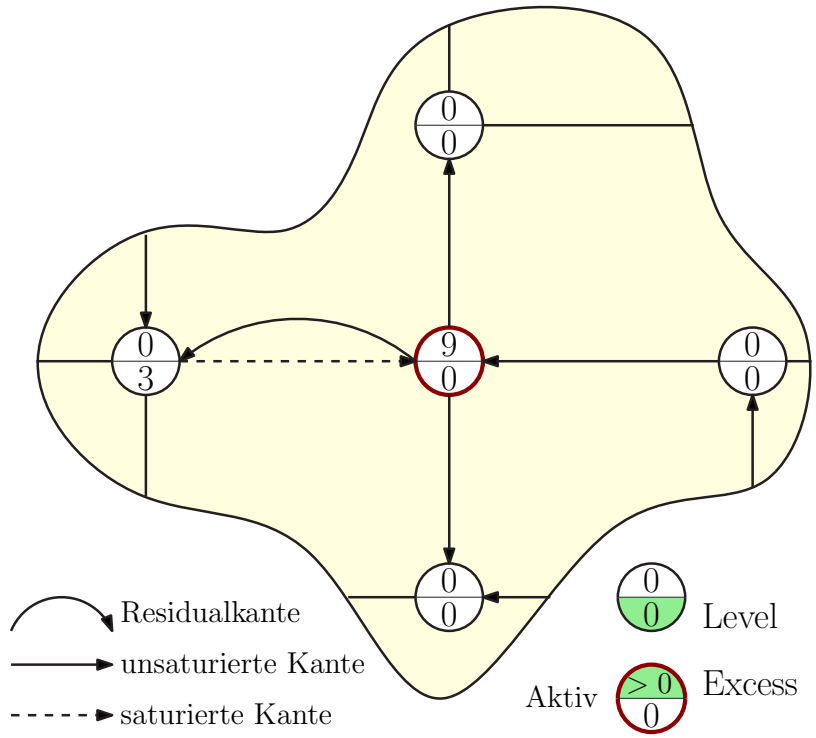
allgemeiner Ablauf

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

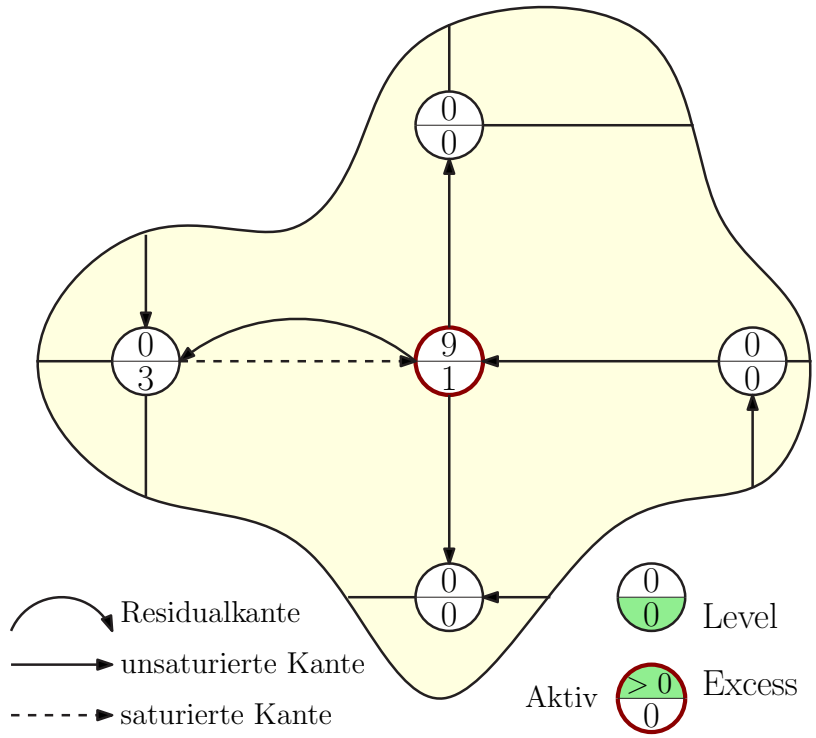
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

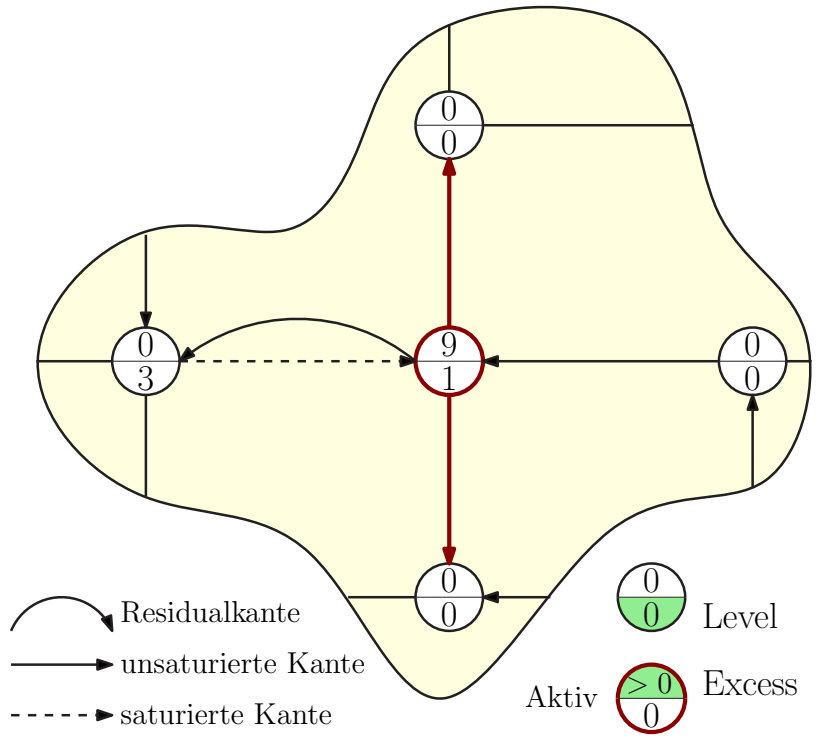
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

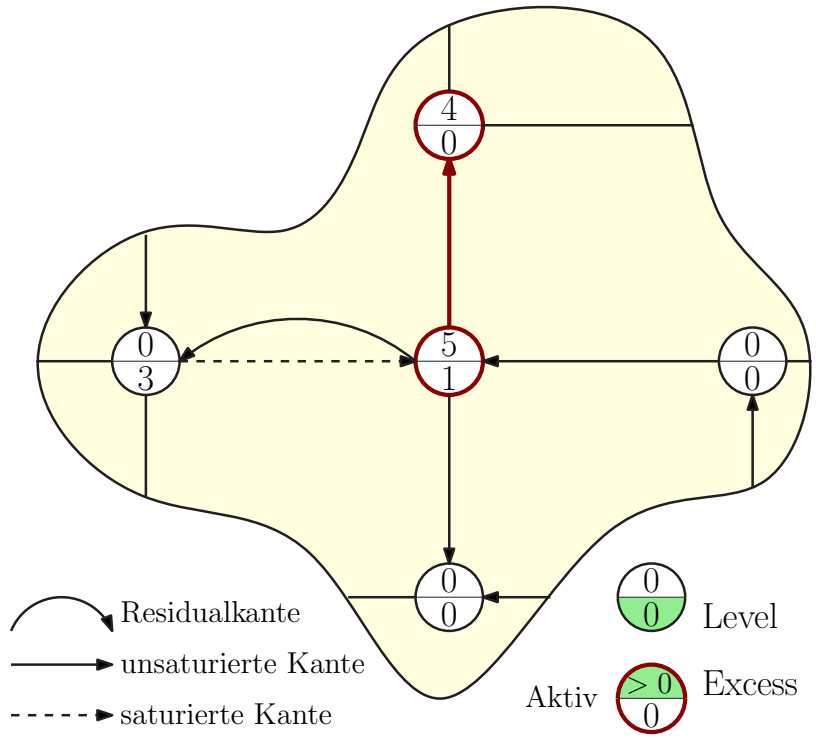
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

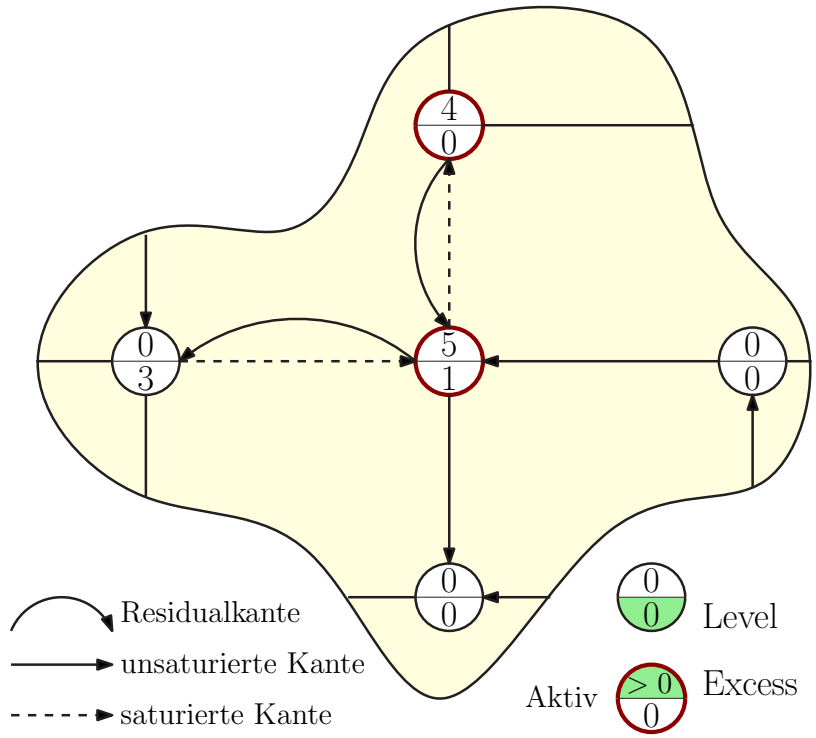
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

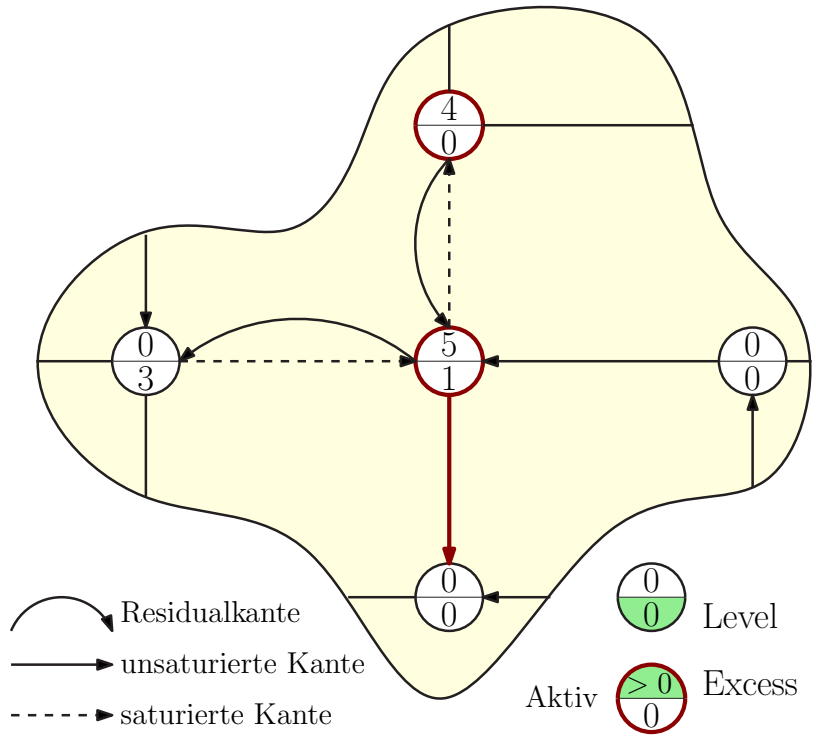
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

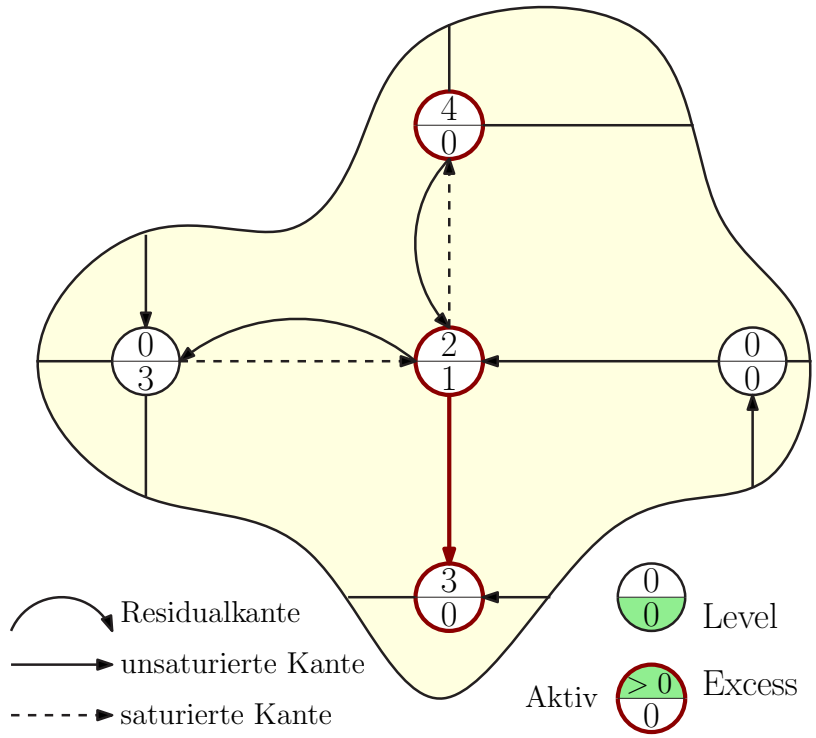
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

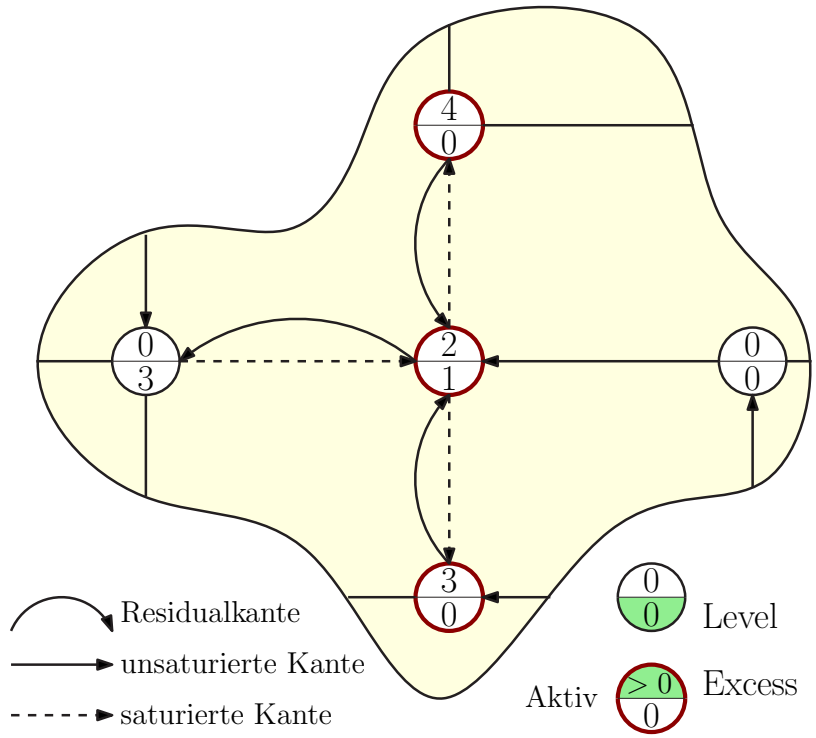
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

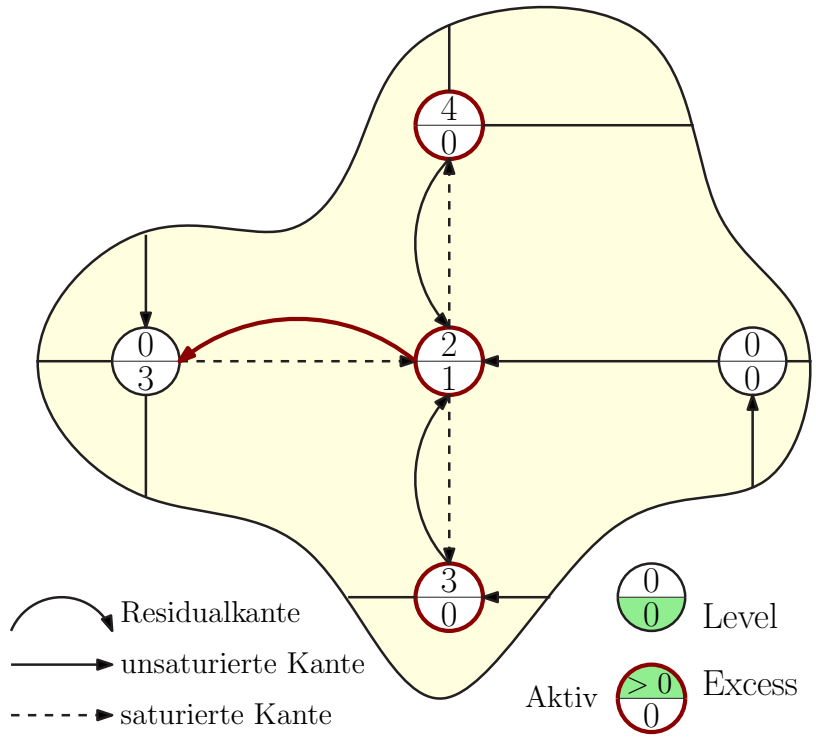
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

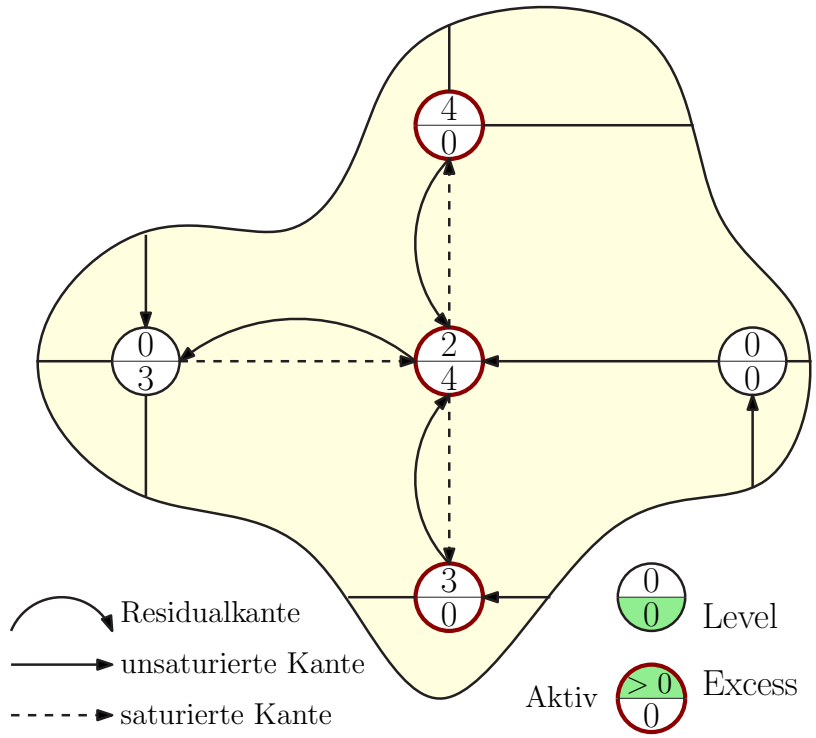
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



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!

preflow-push Algorithmus

Heuristiken

Relabeling

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

Knotenauswahl

- *two-phase approach*

preflow-push Algorithmus

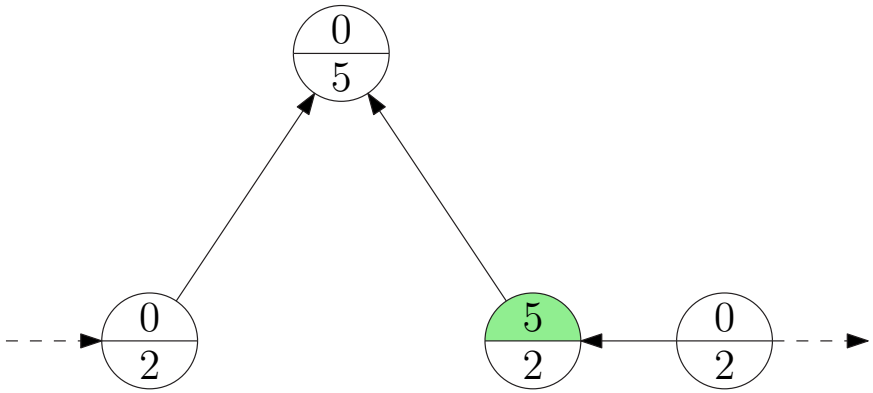
Heuristiken

- *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

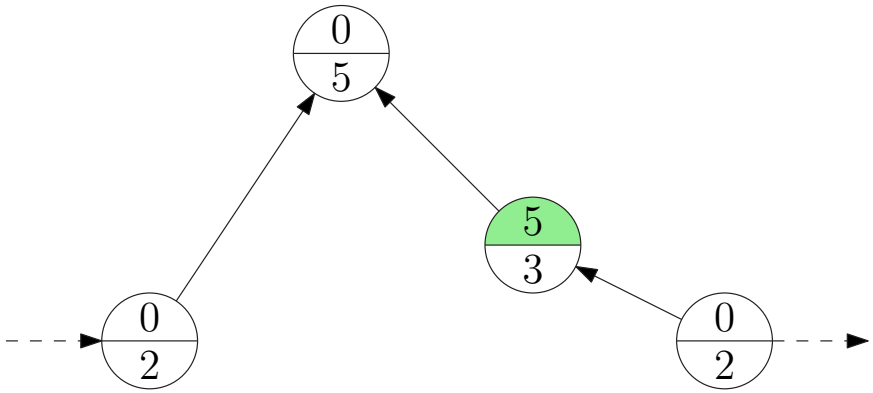
Heuristiken

- *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

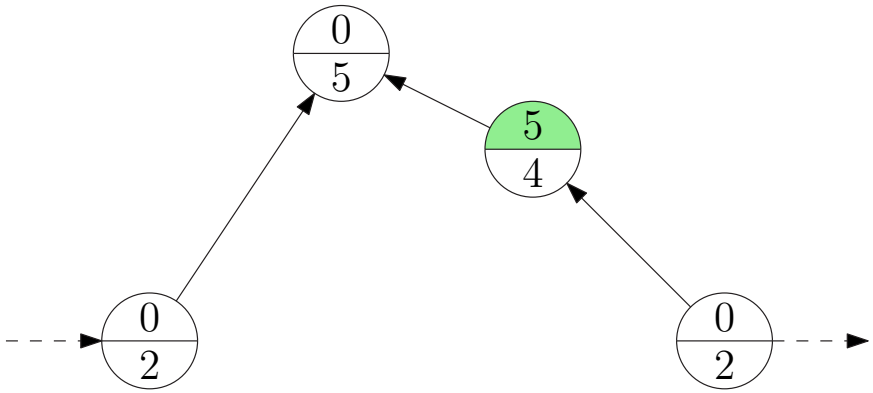
Heuristiken

- *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

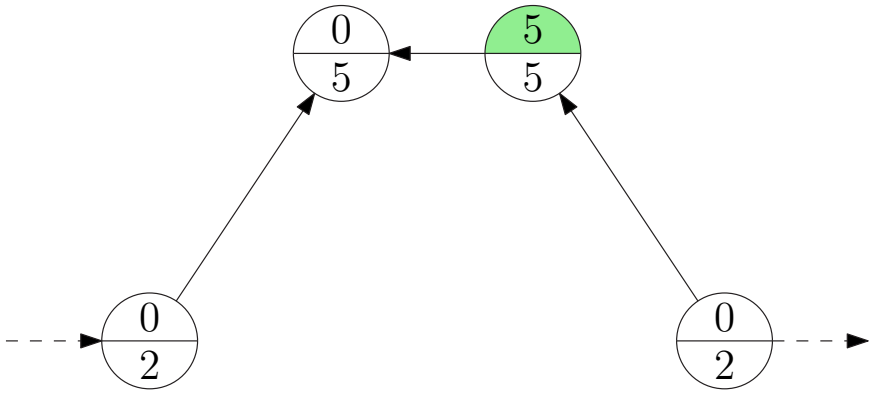
Heuristiken

- *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

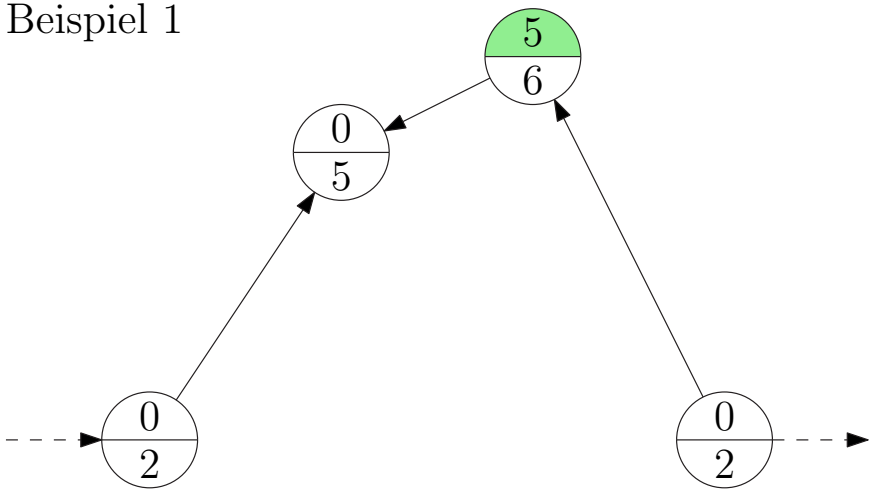
Heuristiken

- *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

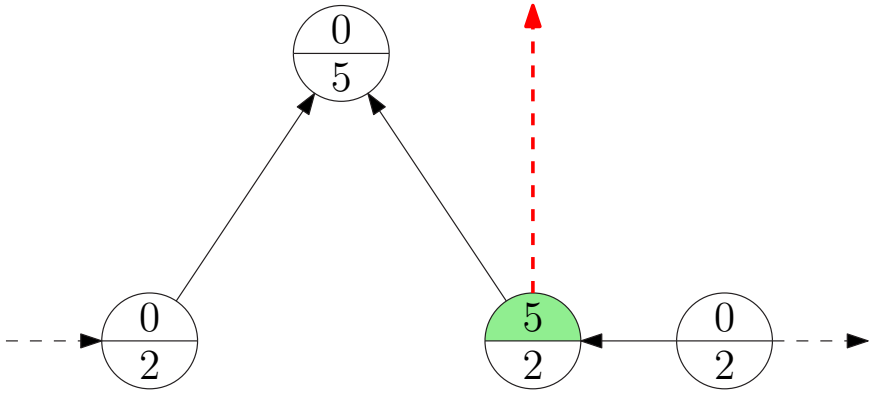
Heuristiken

- *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

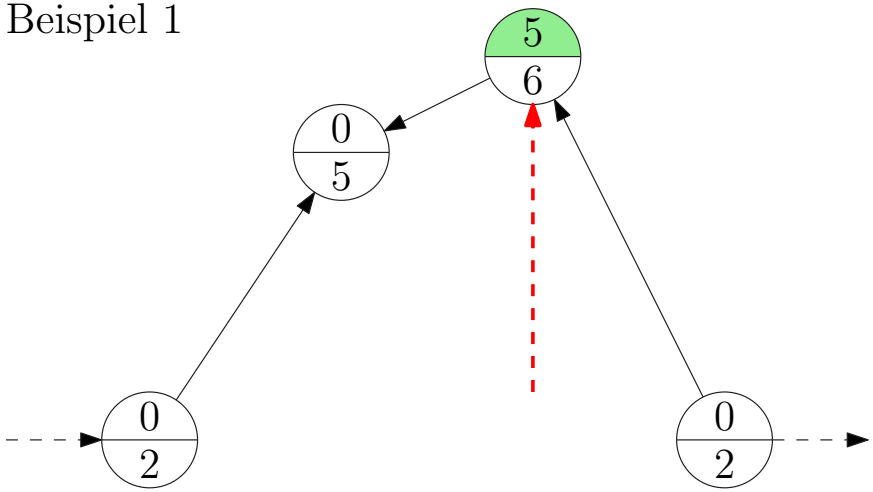
Heuristiken

- *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

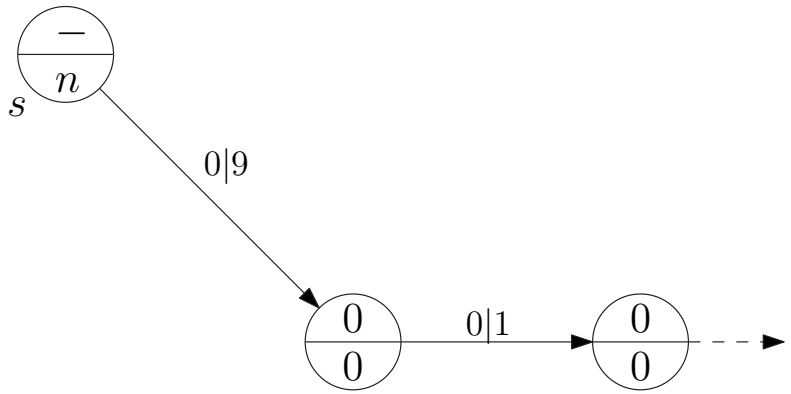
Heuristiken

- *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

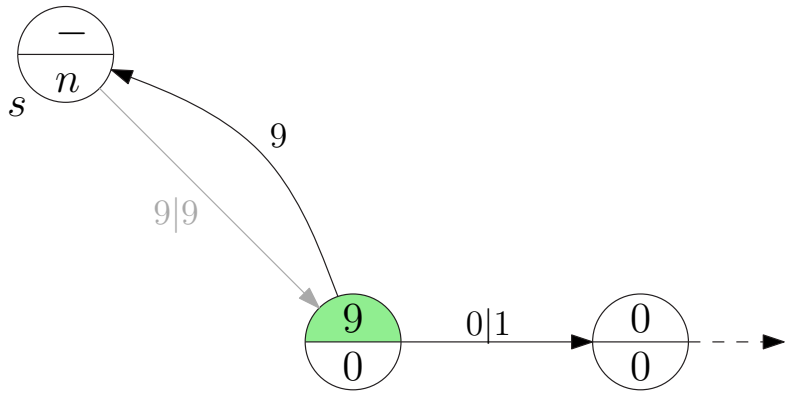
Heuristiken

- *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

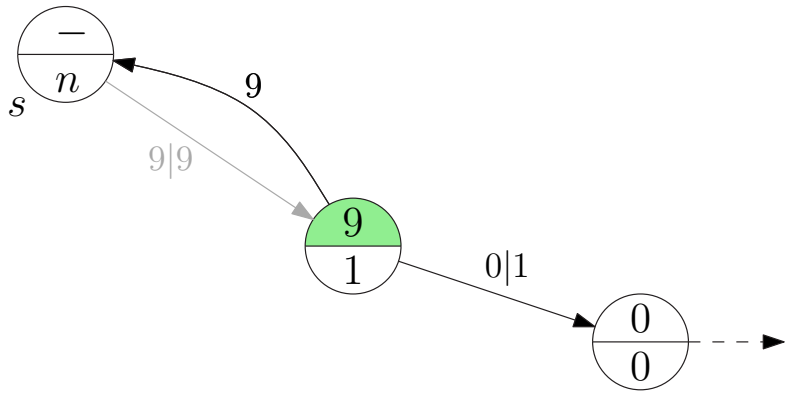
Heuristiken

- *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

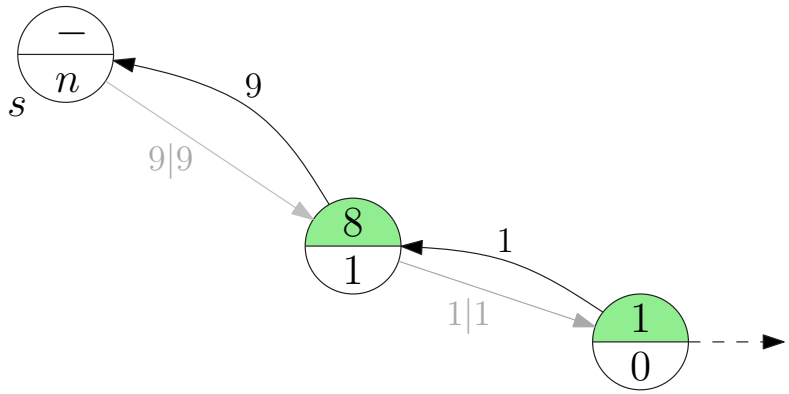
Heuristiken

- *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

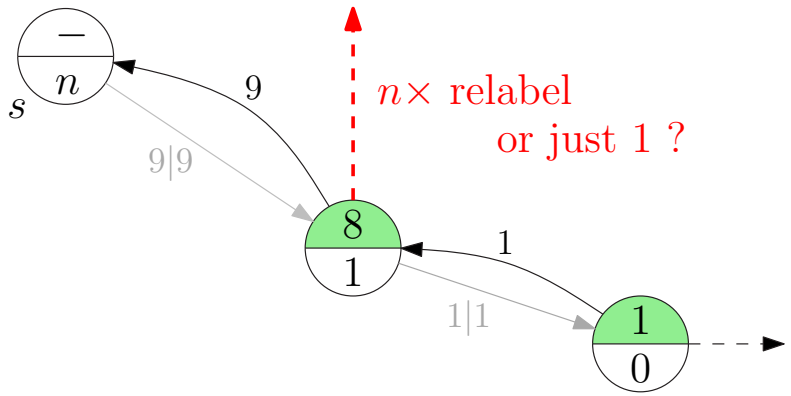
Heuristiken

- *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

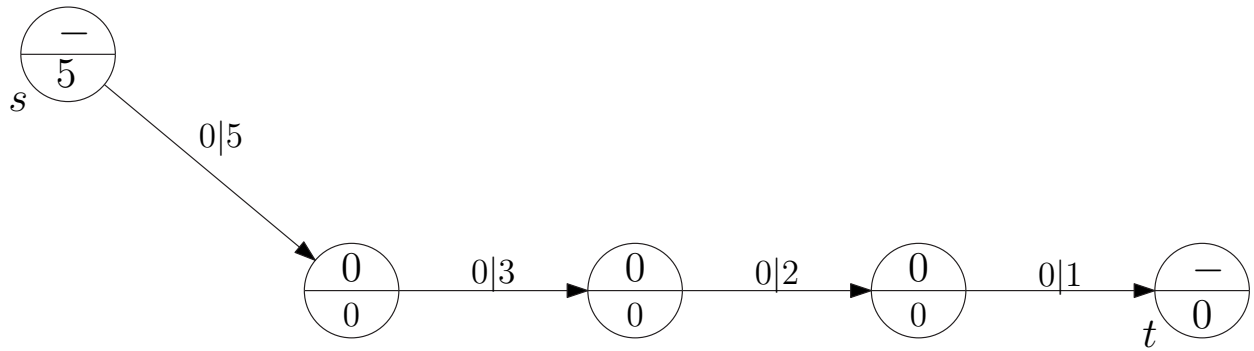
Heuristiken

- *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

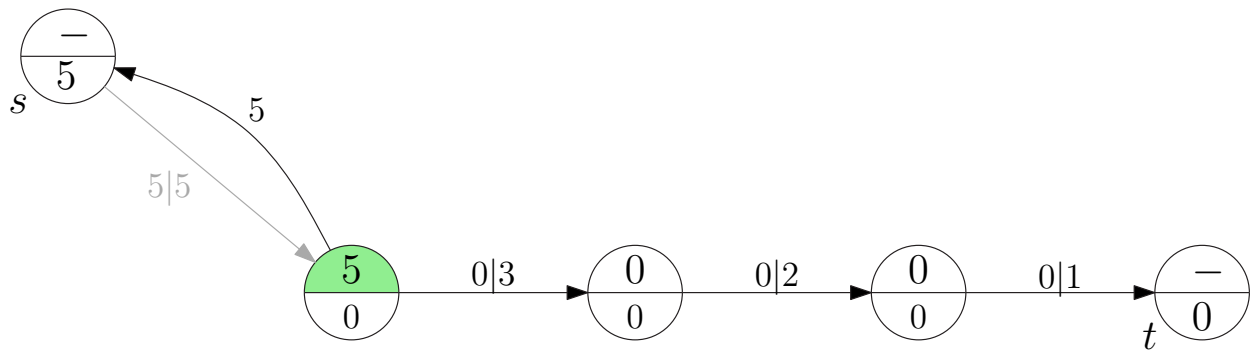
Heuristiken

- *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

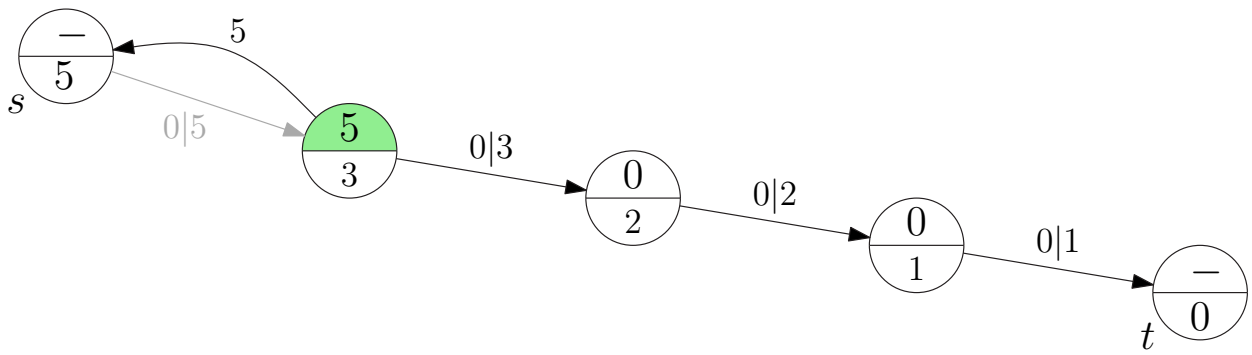
Heuristiken

- *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

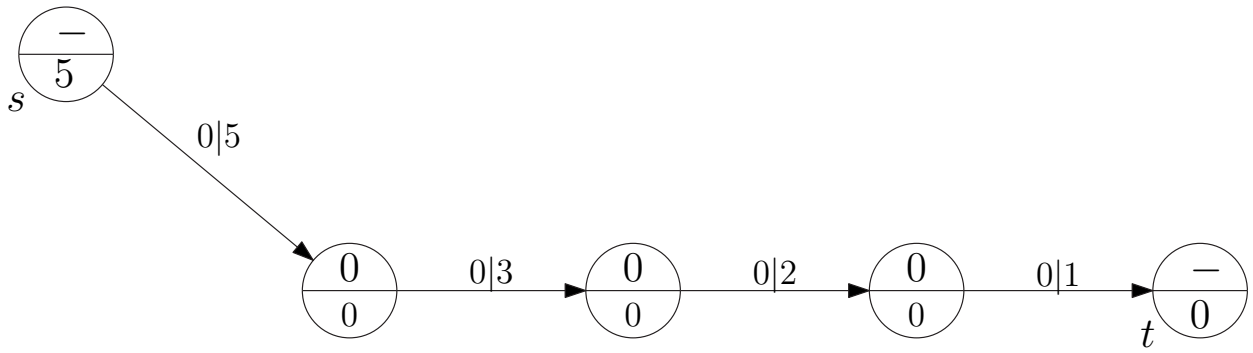
Heuristiken

- *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

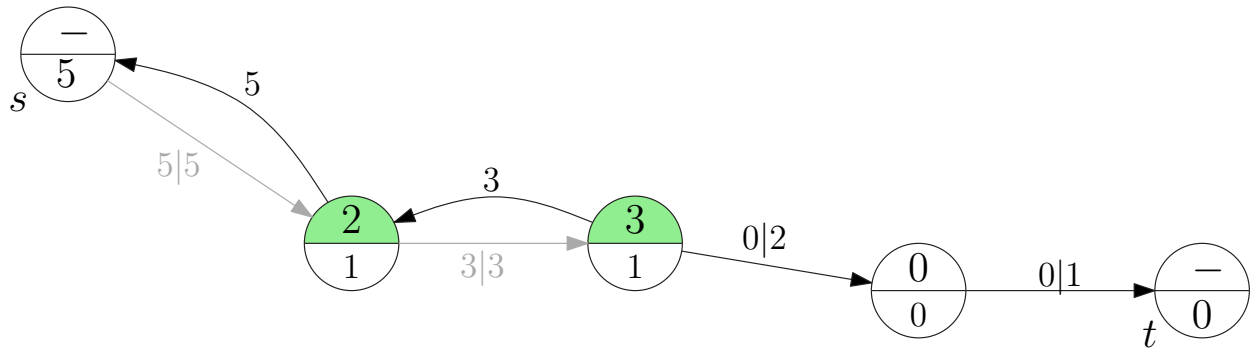
Heuristiken

- *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

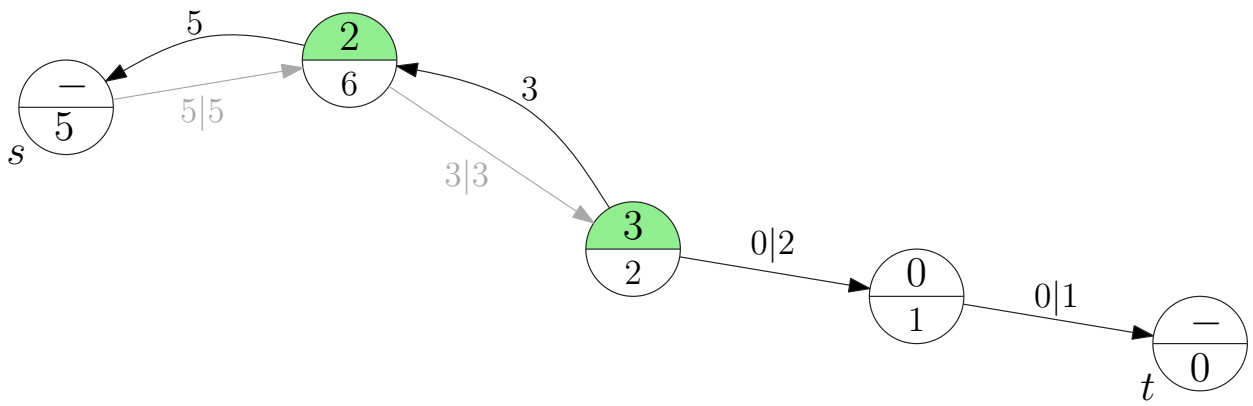
Heuristiken

- *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



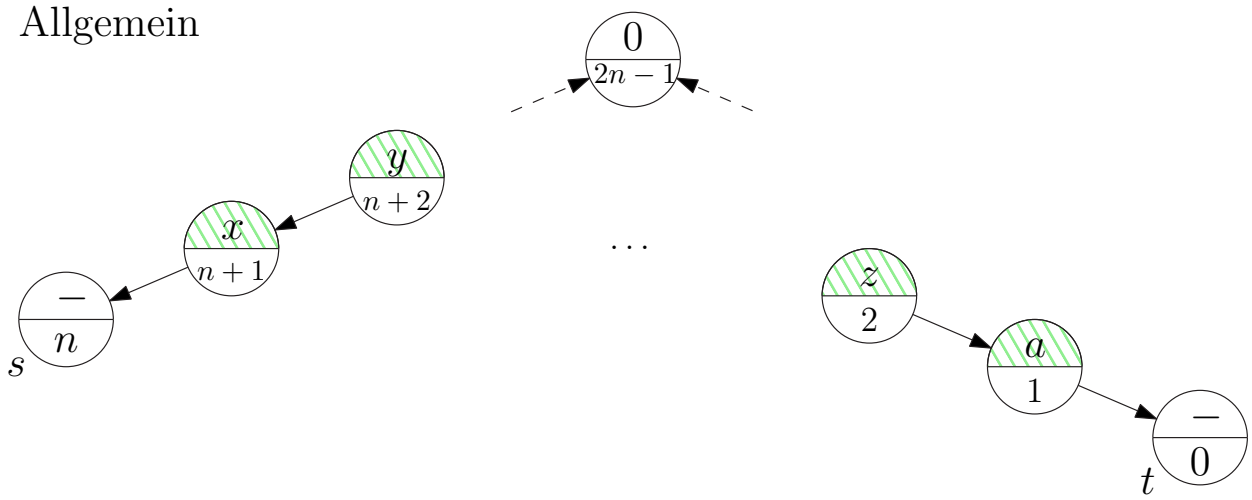
preflow-push Algorithmus

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



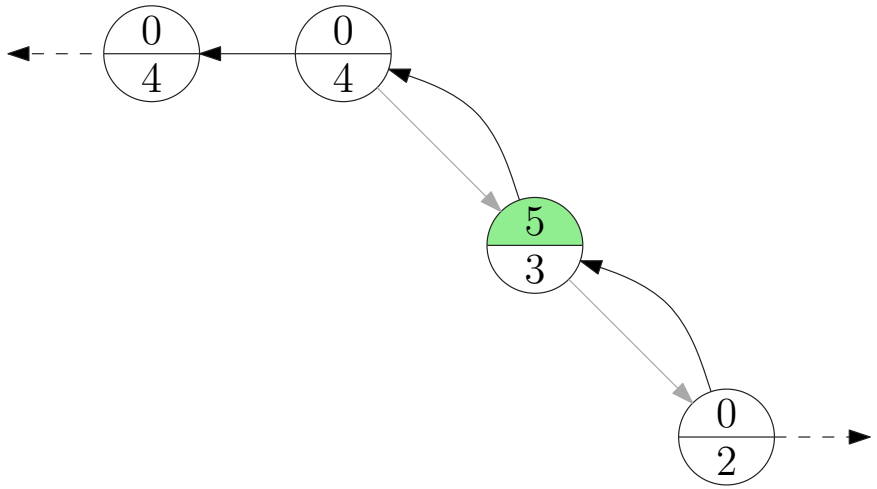
preflow-push Algorithmus

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



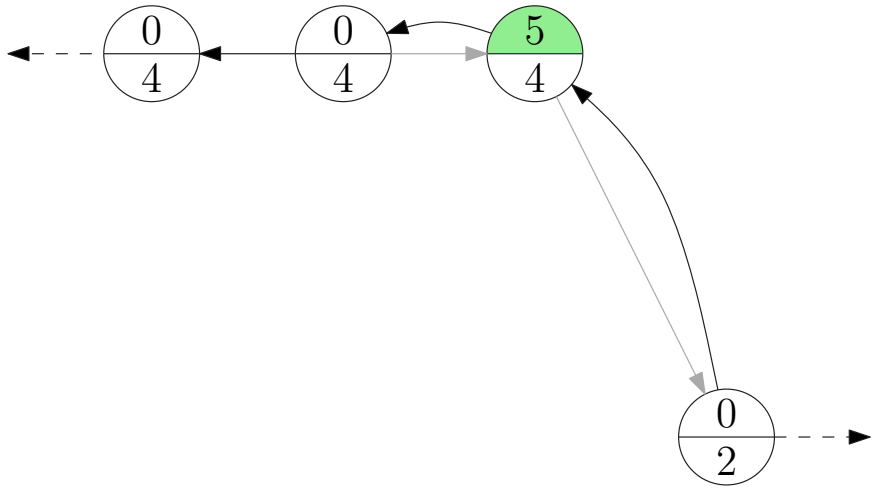
preflow-push Algorithmus

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



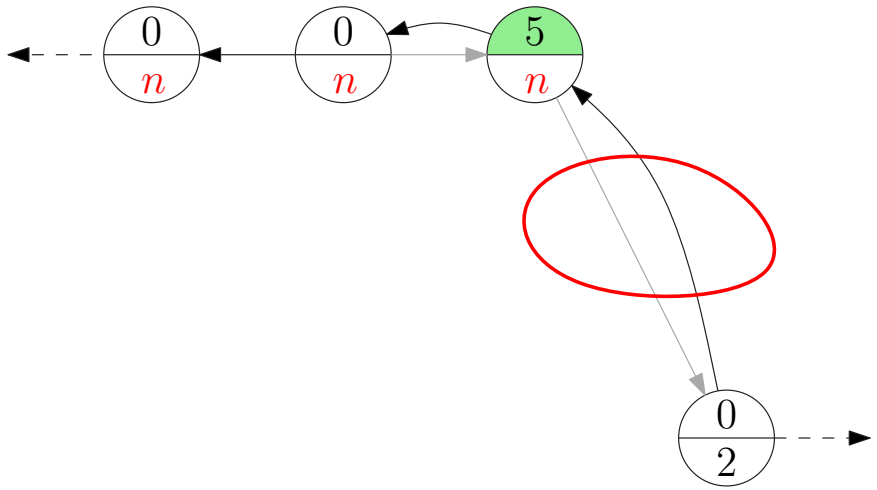
preflow-push Algorithmus

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



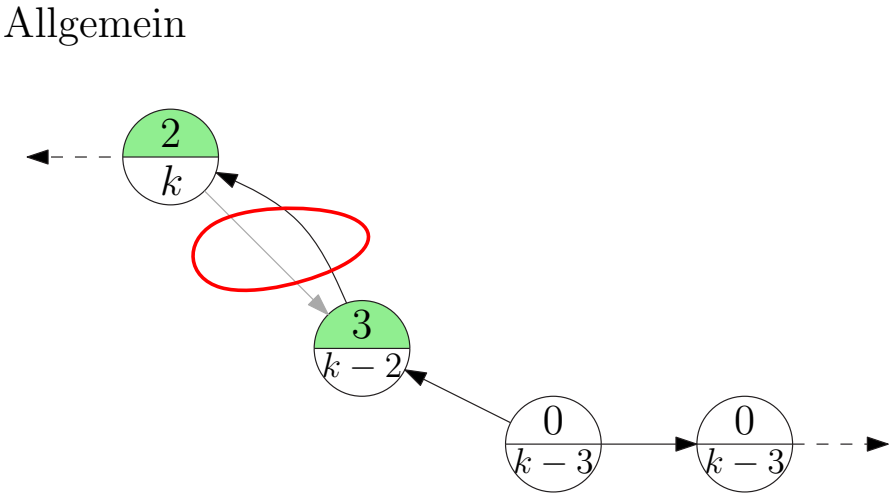
preflow-push Algorithmus

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



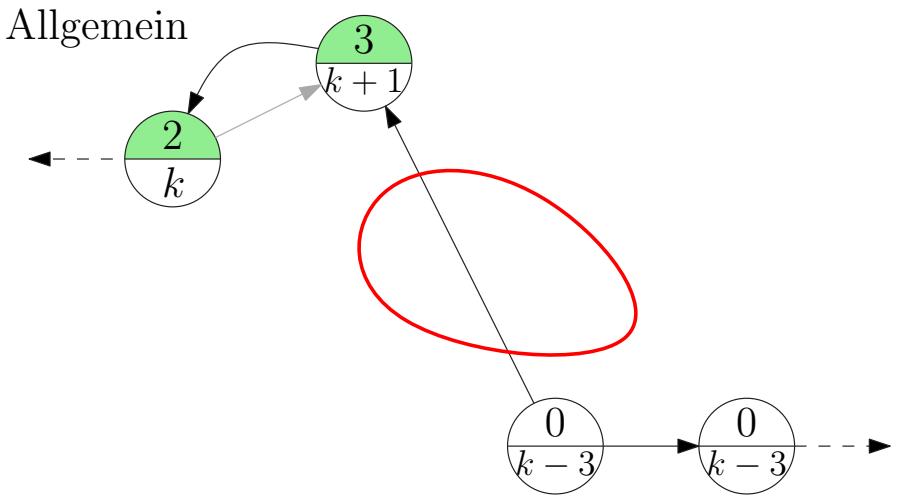
preflow-push Algorithmus

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



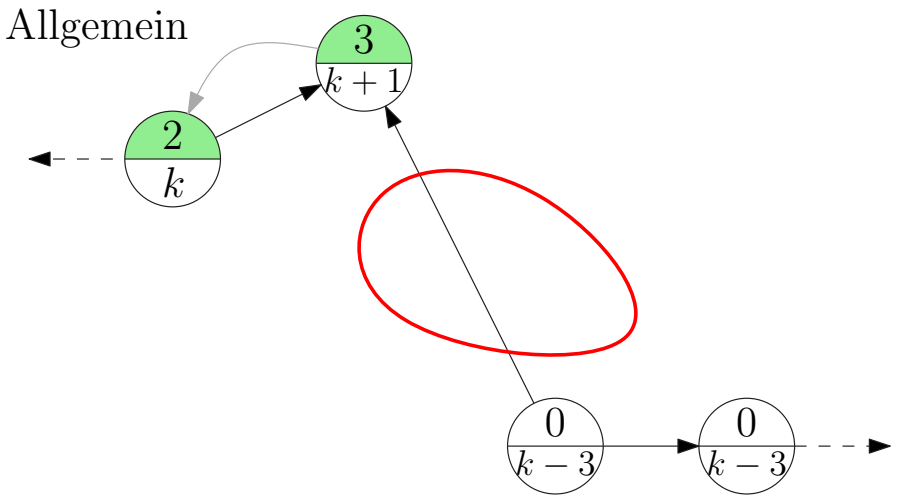
preflow-push Algorithmus

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



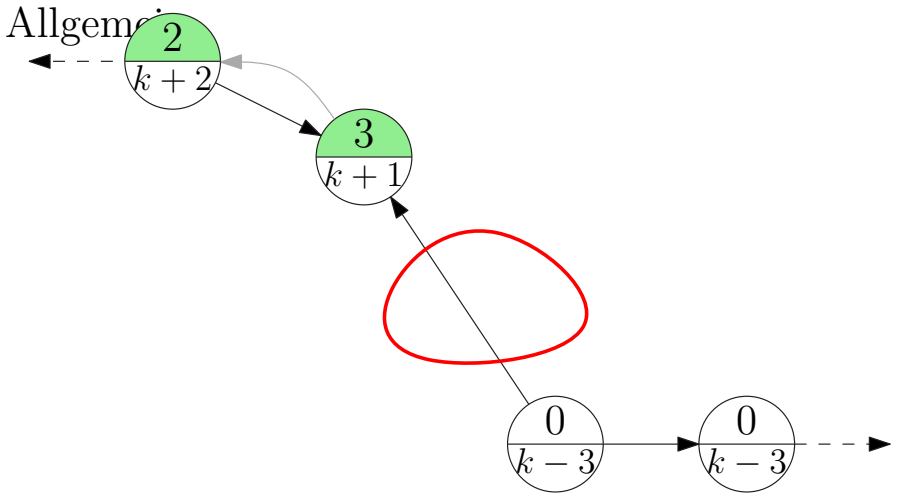
preflow-push Algorithmus

Heuristiken

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

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

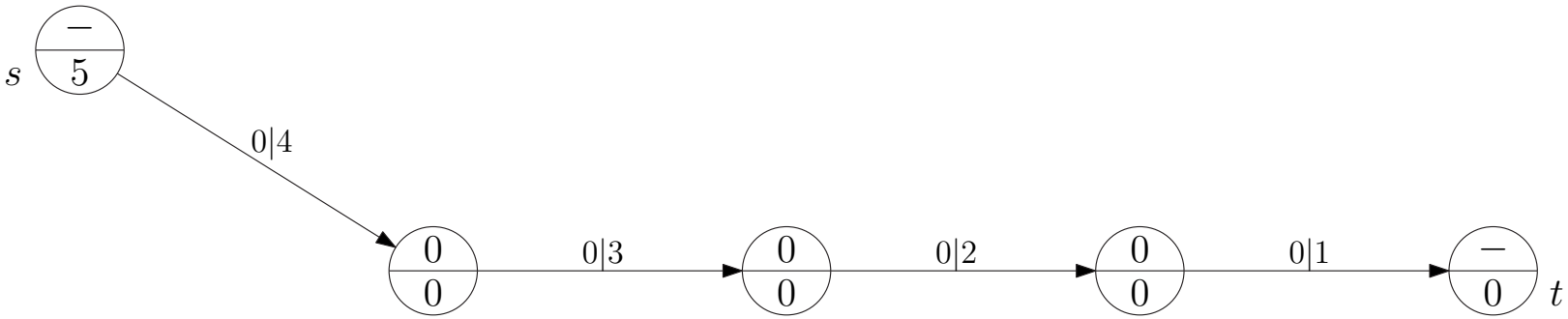
Lücke kann nie mehr überwunden werden, Fluss nur noch zurück zu s



preflow-push Algorithmus

Heuristiken

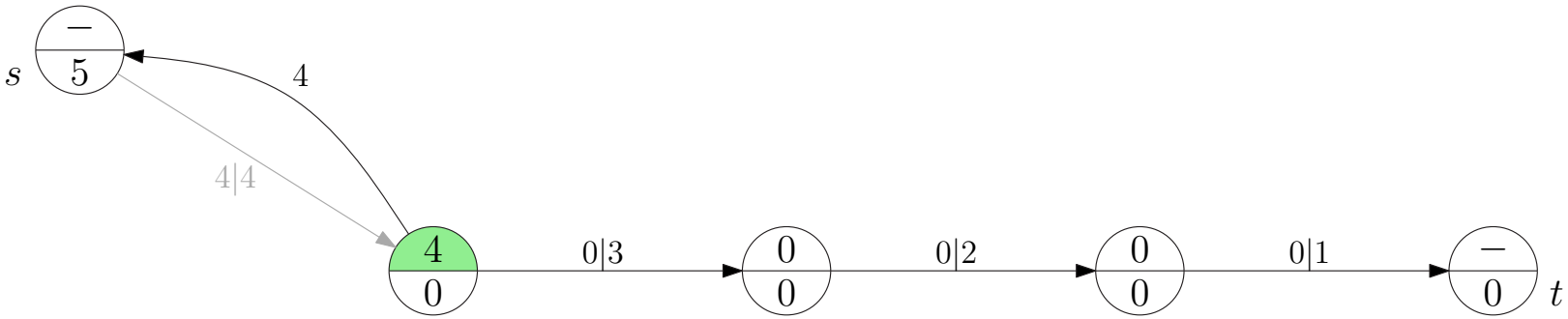
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

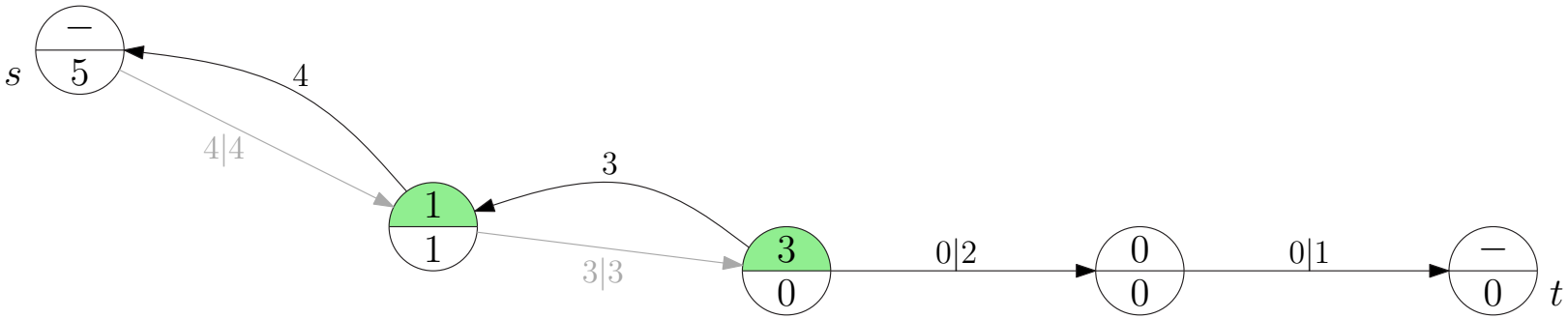
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

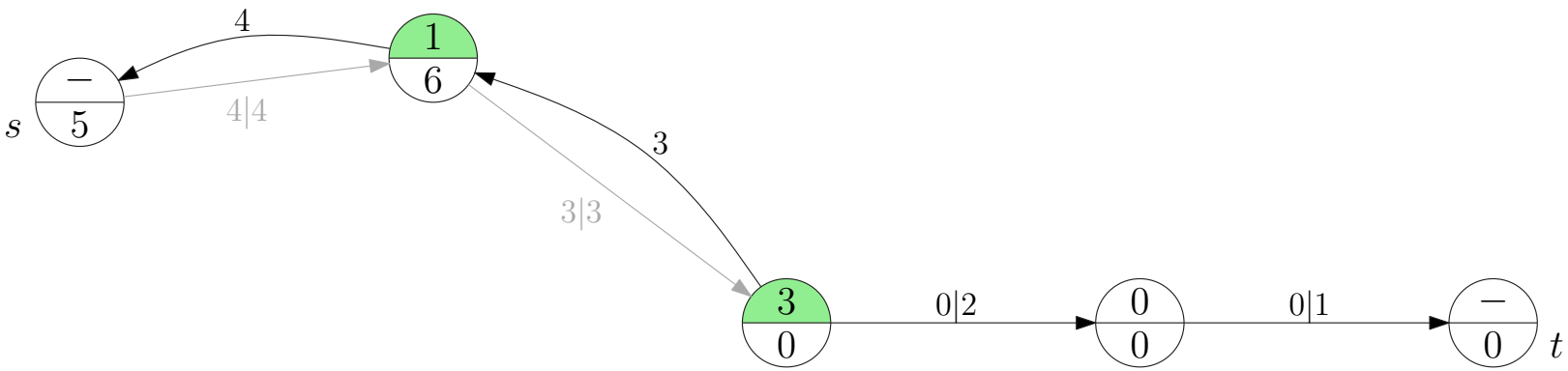
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

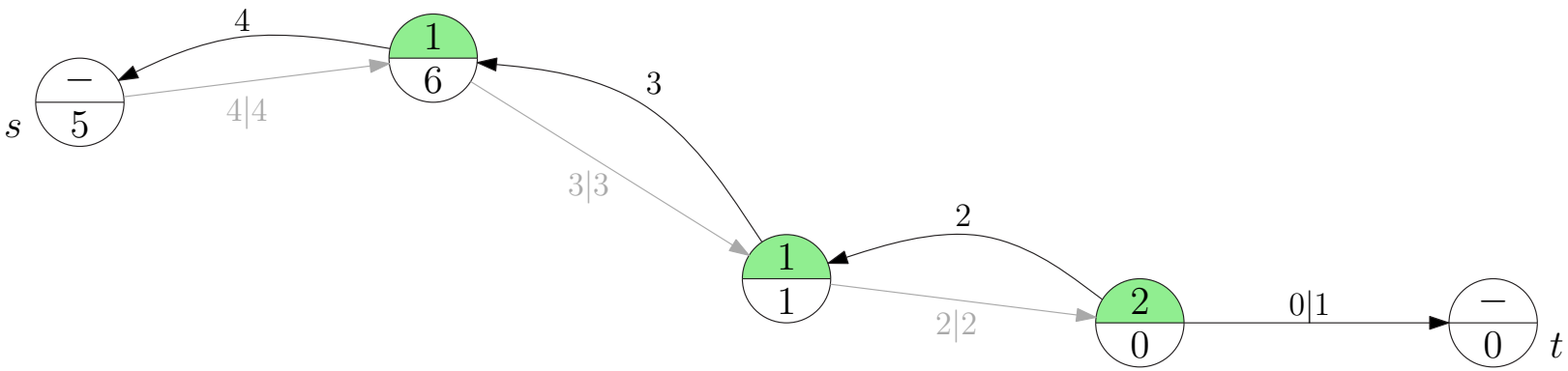
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

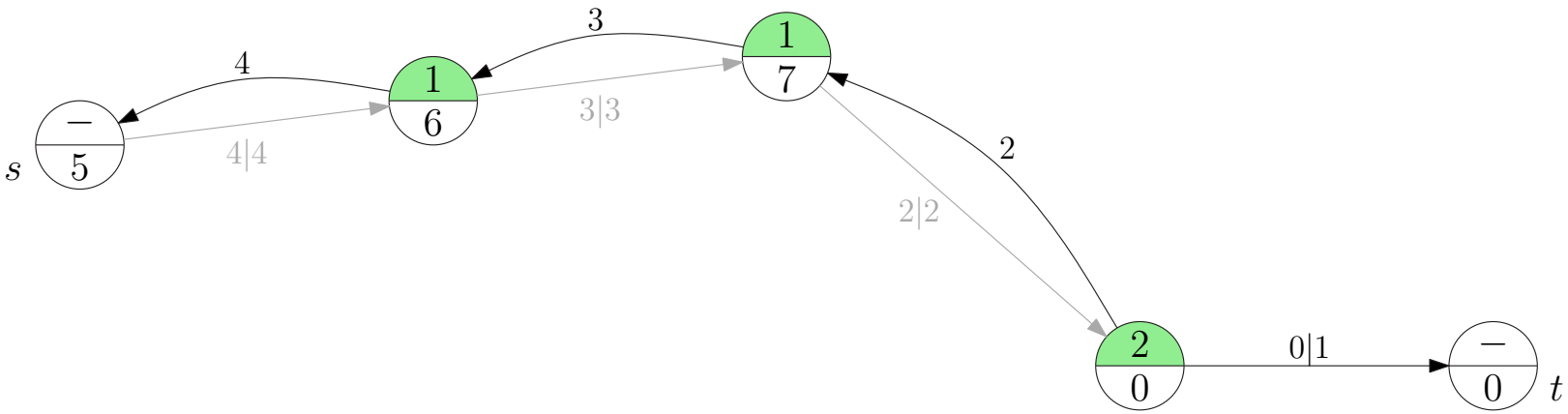
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

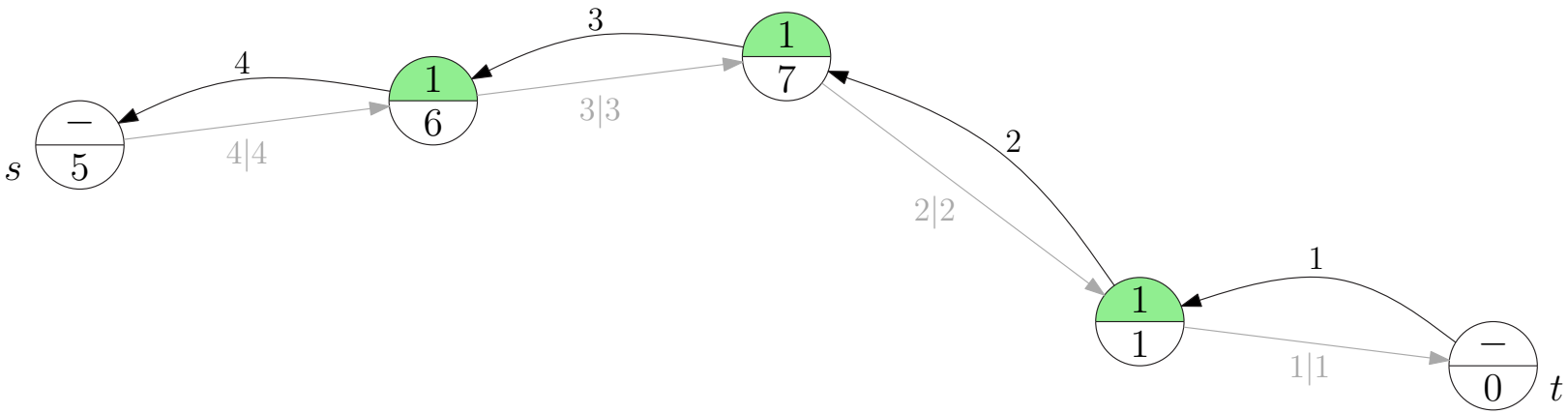
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

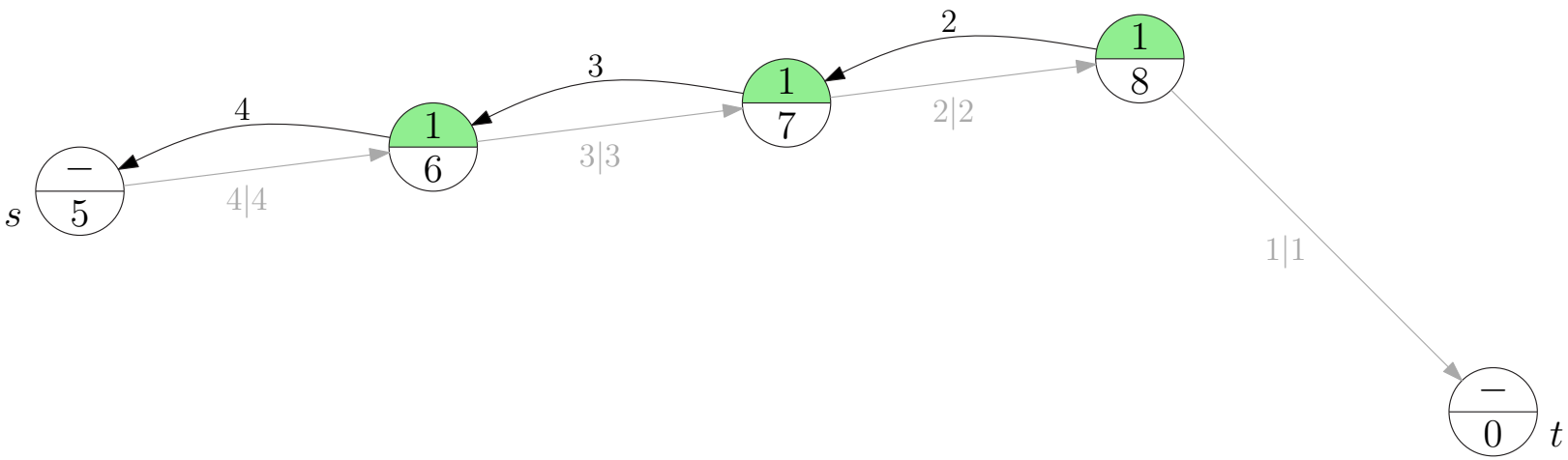
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

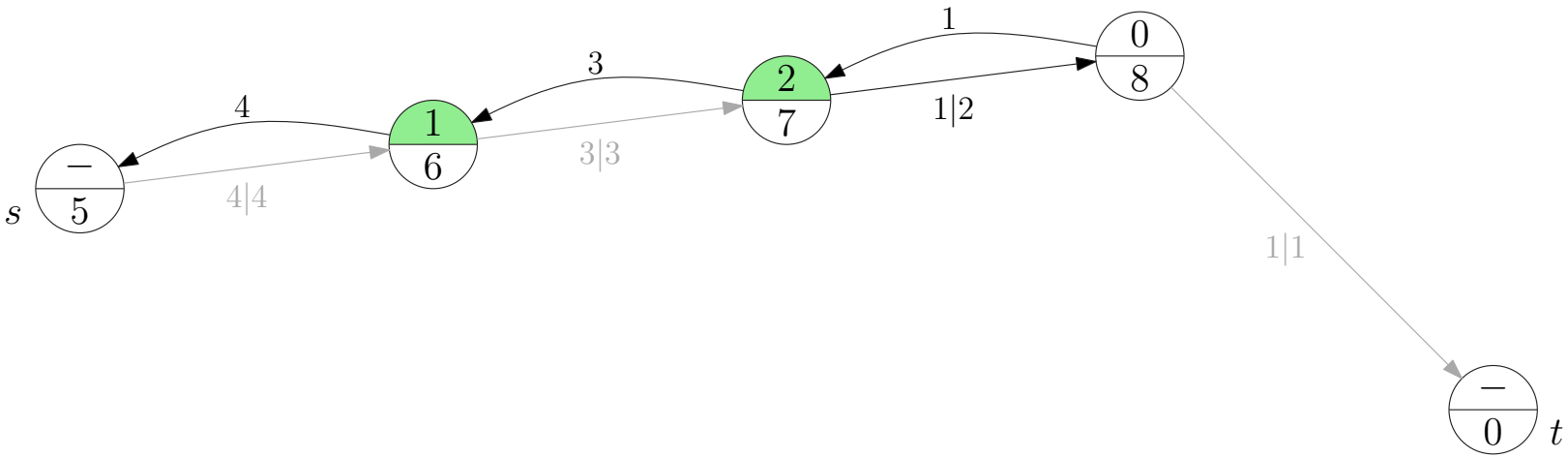
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

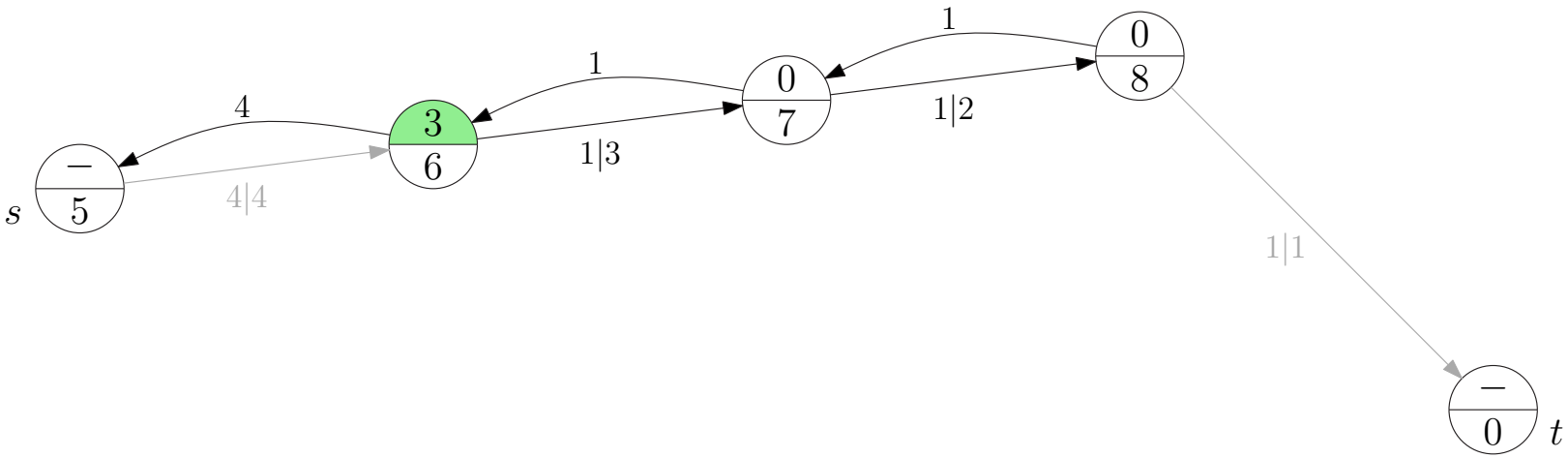
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

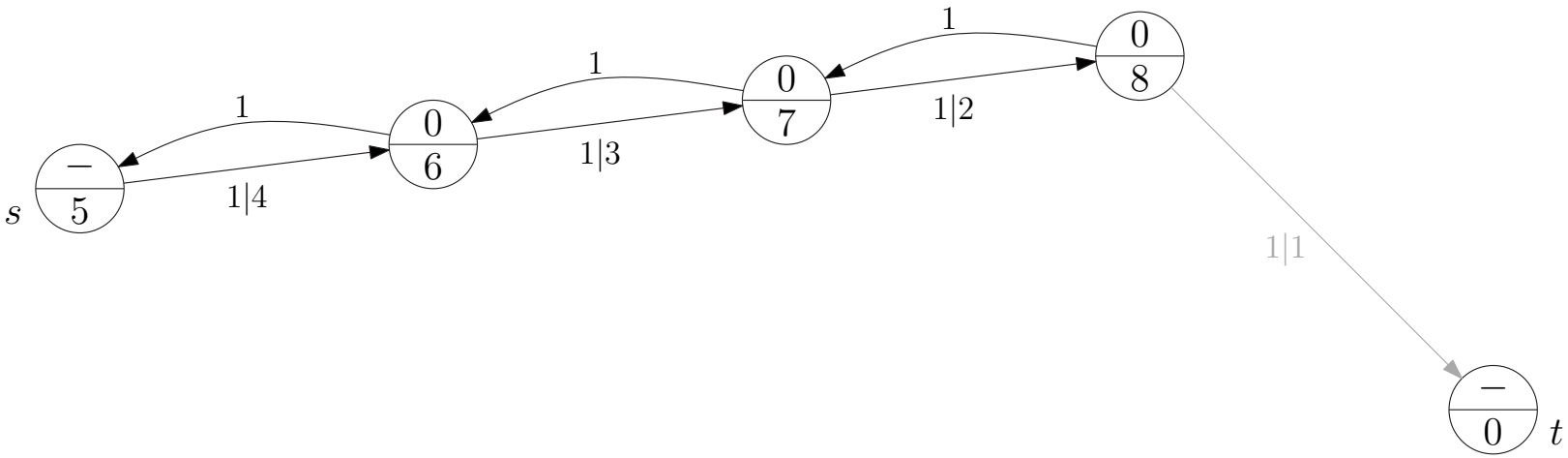
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



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



Randomisierte Algorithmen

Randomisierte Algorithmen

Übersicht

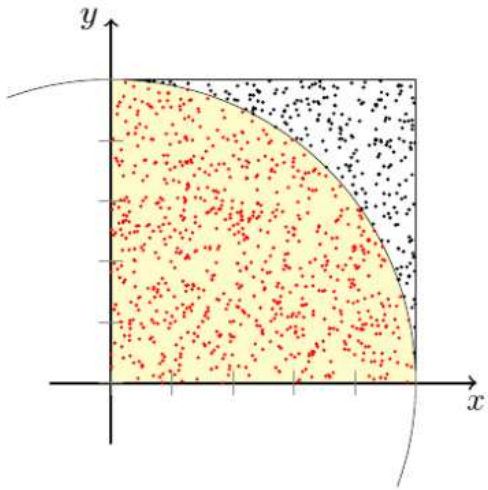
- *Las Vegas Algorithmus*
  - immer korrekte/optimale Lösung
  - Laufzeit ist Zufallsvariable
    $\rightarrow$ erwartete Laufzeit $\mathbb{E}[T]$
  - *Bsp.:* Quicksort

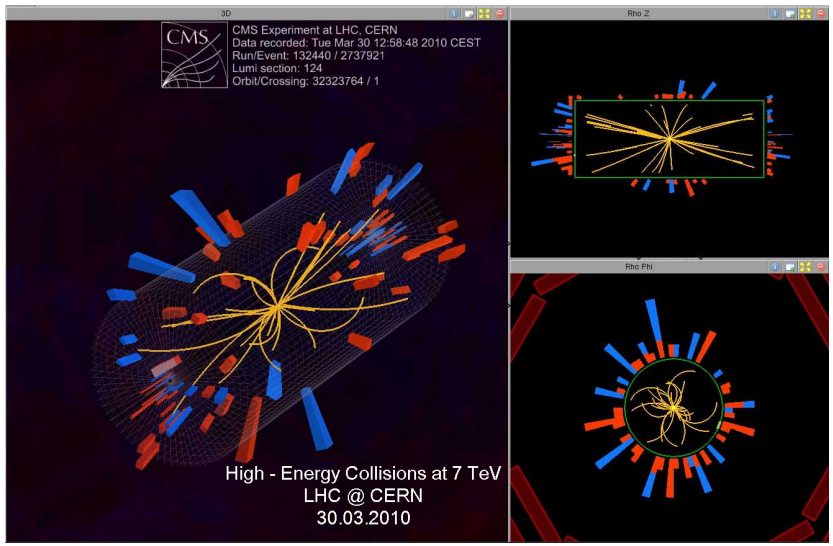
- *Monte Carlo Algorithmus*
  - falsche/suboptimale Lösung möglich
    $\rightarrow$ mit Wahrscheinlichkeit p
  - deterministische Laufzeit
  - *Bsp.:* Miller-Rabin Primzahltest

Randomisierte Algorithmen

Monte Carlo Simulation

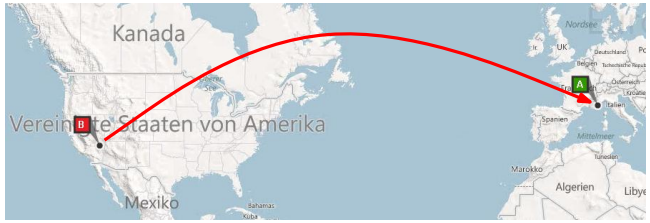
- *Monte Carlo Simulation*
  - nicht zu verwechseln mit *Monte Carlo Algorithmus*!
  - Zufallsexperimente sehr oft ausführen Gesetz der großen Zahlen
  - je länger / öfter ausgeführt, desto bessere Ergebnisse
    - Alternative zur analytischen Lösung
    - 3D-Rendering
    - Nachbildung komplexer Prozesse





Randomisierte Algorithmen

Las Vegas $\rightarrow$ Monte Carlo



geg: Las Vegas Algorithmus mit erwarteter Laufzeit $\mathbb{E}[T] = f(n)$
ges: Monte Carlo Algorithmus mit Laufzeit $\mathcal{O}(f(n))$, Fehlerrate p

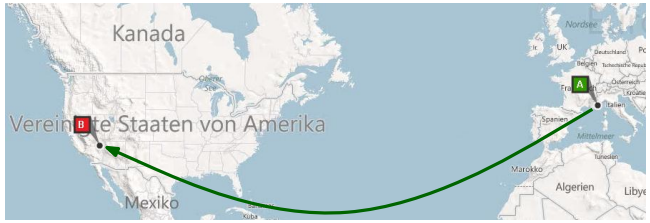
Idee: Abbruch nach Zeit $\alpha f(n)$

- Ausgabe FALSCH, wenn Algorithmus abgebrochen wurde
- $\mathbb{P}[T > \alpha f(n)] \leq 1/\alpha$ Markov Ungleichung

$\rightarrow$ Monte Carlo Algorithmus mit Laufzeit $\alpha f(n)$ und Fehlerrate $p = 1/\alpha$

Randomisierte Algorithmen

Monte Carlo → Las Vegas



geg: Monte Carlo Algorithmus mit Laufzeit $\mathcal{O}(f(n))$, Fehlerrate p, Korrektheit in $\mathcal{O}(g(n))$ prüfbar
ges: Las Vegas Algorithmus mit erwarteter Laufzeit $\mathbb{E}[T]$

Idee: Wiederhole MC bis korrektes Ergebnis gefunden

- Laufzeit $T \leq i \cdot \mathcal{O}(f(n) + g(n))$ i Schritte benötigt
- $\mathbb{E}[T] \leq \mathbb{E}[i] \cdot \mathcal{O}(f(n) + g(n))$
  - $\mathbb{E}[i] = \sum_{k=1}^{\infty} k \cdot p^{k-1} \cdot (1-p) = \frac{1}{1-p}$
    nach $\sum_{k=1}^{\infty} kx^k = x/(1-x)^2,\ x < 1$

→ Las Vegas Algorithmus mit erwarteter Laufzeit $\mathbb{E}[T] \leq \frac{\mathcal{O}(f(n)+g(n))}{1-p}$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

Aufgabe: Überprüfe, ob $X \cdot Y = Z$ für Matrizen X, Y, Z

- deterministisch:
  - berechne $X \cdot Y$ und vergleiche mit Z

  $\rightarrow$ Laufzeit $\mathcal{O}(n^3)$ (naiv), $\mathcal{O}(n^{2.37})$ (best)

- randomisiert:
  - Wähle 0-1 Vektor $r = (r_1, \ldots, r_n)$ zufällig
  - Wenn $X(Yr) = Zr$ dann KORREKT, sonst FALSCH

  $\rightarrow$ Laufzeit $\mathcal{O}(n^2)$

Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

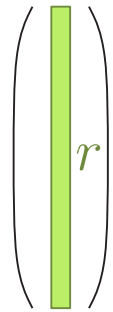
Aufgabe: Überprüfe, ob $X \cdot Y = Z$ für Matrizen X, Y, Z

- deterministisch:
  - berechne $X \cdot Y$ und vergleiche mit Z

  $\rightarrow$ Laufzeit $\mathcal{O}(n^3)$ (naiv), $\mathcal{O}(n^{2.37})$ (best)

- randomisiert:
  - Wähle 0-1 Vektor $r = (r_1, \ldots, r_n)$ zufällig
  - Wenn $X(Yr) = Zr$ dann KORREKT, sonst FALSCH

  $\rightarrow$ Laufzeit $\mathcal{O}(n^2)$



Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

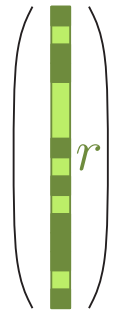
Aufgabe: Überprüfe, ob $X \cdot Y = Z$ für Matrizen X, Y, Z

- deterministisch:
  - berechne $X \cdot Y$ und vergleiche mit Z

  $\rightarrow$ Laufzeit $\mathcal{O}(n^3)$ (naiv), $\mathcal{O}(n^{2.37})$ (best)

- randomisiert:
  - Wähle 0-1 Vektor $r = (r_1, \ldots, r_n)$ zufällig
  - Wenn $X(Yr) = Zr$ dann KORREKT, sonst FALSCH

  $\rightarrow$ Laufzeit $\mathcal{O}(n^2)$



Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

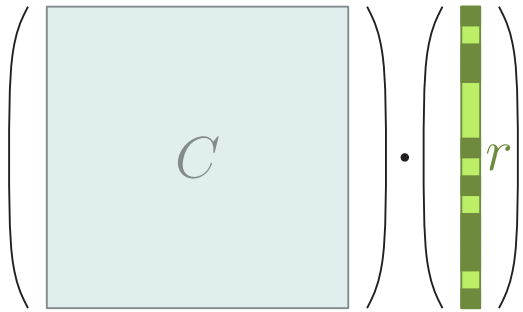
Aufgabe: Überprüfe, ob $X \cdot Y = Z$ für Matrizen X, Y, Z

- deterministisch:
  - berechne $X \cdot Y$ und vergleiche mit Z

  $\rightarrow$ Laufzeit $\mathcal{O}(n^3)$ (naiv), $\mathcal{O}(n^{2.37})$ (best)

- randomisiert:
  - Wähle 0-1 Vektor $r = (r_1, \ldots, r_n)$ zufällig
  - Wenn $X(Yr) = Zr$ dann KORREKT, sonst FALSCH

  $\rightarrow$ Laufzeit $\mathcal{O}(n^2)$



Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

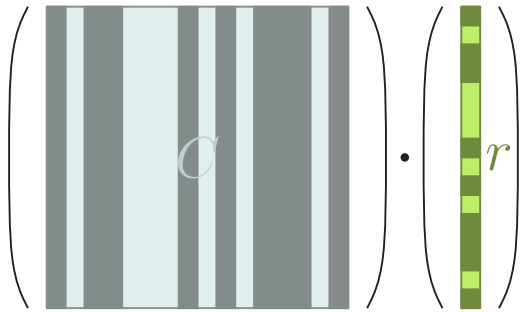
Aufgabe: Überprüfe, ob $X \cdot Y = Z$ für Matrizen X, Y, Z

- deterministisch:
  - berechne $X \cdot Y$ und vergleiche mit Z

  $\rightarrow$ Laufzeit $\mathcal{O}(n^3)$ (naiv), $\mathcal{O}(n^{2.37})$ (best)

- randomisiert:
  - Wähle 0-1 Vektor $r = (r_1, \ldots, r_n)$ zufällig
  - Wenn $X(Yr) = Zr$ dann KORREKT, sonst FALSCH

  $\rightarrow$ Laufzeit $\mathcal{O}(n^2)$



Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Definitionen:

- $A := XY$, $B := Z$

Annahme: $A \neq B$; Wann ist dann $Ar = Br$?

- $\exists i, j : A_{i,j} \neq B_{i,j}$
- Sei $a := A_i$ und $b := B_i$ A_i: i'te Zeile von A
  $\alpha := \sum_{k \neq j} a_k r_k$ und $\beta := \sum_{k \neq j} b_k r_k$
  $\Rightarrow ar = \alpha + a_j r_j$ und $br = \beta + b_j r_j$
  $\Rightarrow ar - br = (\alpha - \beta) + (a_j - b_j) r_j$
- Annahme: Werte für $r_{1 \ldots j-1, j+1 \ldots n}$ bereits festgelegt
  $\rightarrow$ noch 2 Möglichkeiten für r_j, Gleichheit bei maximal einer
  $\Rightarrow \mathbf{Pr}[ar - br = 0] = \mathbf{Pr}[r_j = \frac{\beta - \alpha}{a_j - b_j}] \leq \frac{1}{2}$

$\rightarrow$ Fehlerrate $p \leq 0.5$

Randomisierte Algorithmen

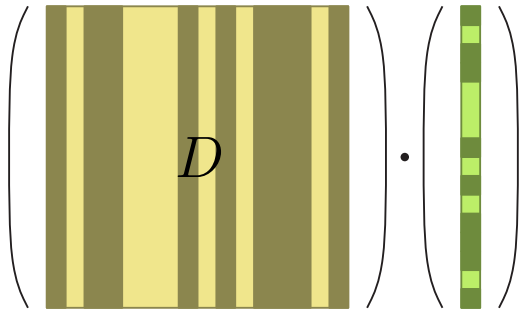
Matrix-Matrix Multiplikation

Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Definitionen:

- $A := XY$, $B := Z$

Annahme: $A \neq B$; Wann ist dann $Ar = Br$?



- $\exists i, j : A_{i,j} \neq B_{i,j}$
- Sei $a := A_i$ und $b := B_i$ A_i: i'te Zeile von A
  $\alpha := \sum_{k \neq j} a_k r_k$ und $\beta := \sum_{k \neq j} b_k r_k$
  $\Rightarrow ar = \alpha + a_j r_j$ und $br = \beta + b_j r_j$
  $\Rightarrow ar - br = (\alpha - \beta) + (a_j - b_j) r_j$
- Annahme: Werte für $r_{1 \ldots j-1, j+1 \ldots n}$ bereits festgelegt
  $\rightarrow$ noch 2 Möglichkeiten für r_j, Gleichheit bei maximal einer
  $\Rightarrow \mathbf{Pr}[ar - br = 0] = \mathbf{Pr}[r_j = \frac{\beta - \alpha}{a_j - b_j}] \leq \frac{1}{2}$

$\rightarrow$ Fehlerrate $p \leq 0.5$

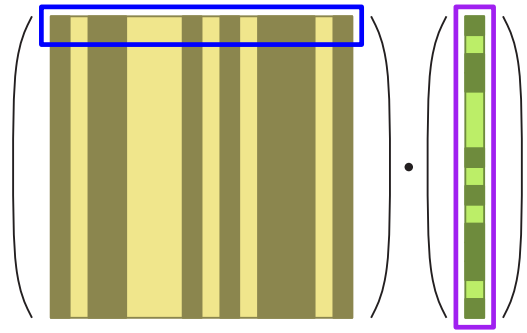
Randomisierte Algorithmen

Matrix-Matrix Multiplikation

Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Definitionen:

- $A := XY$, $B := Z$



Annahme: $A \neq B$; Wann ist dann $Ar = Br$?

- $\exists i, j : A_{i,j} \neq B_{i,j}$
- Sei $a := A_i$ und $b := B_i$ A_i: i'te Zeile von A
  $\alpha := \sum_{k \neq j} a_k r_k$ und $\beta := \sum_{k \neq j} b_k r_k$
  $\Rightarrow ar = \alpha + a_j r_j$ und $br = \beta + b_j r_j$
  $\Rightarrow ar - br = (\alpha - \beta) + (a_j - b_j) r_j$
- Annahme: Werte für $r_{1 \ldots j-1, j+1 \ldots n}$ bereits festgelegt
  $\rightarrow$ noch 2 Möglichkeiten für r_j, Gleichheit bei maximal einer
  $\Rightarrow \mathbf{Pr}[ar - br = 0] = \mathbf{Pr}[r_j = \frac{\beta - \alpha}{a_j - b_j}] \leq \frac{1}{2}$

$\rightarrow$ Fehlerrate $p \leq 0.5$

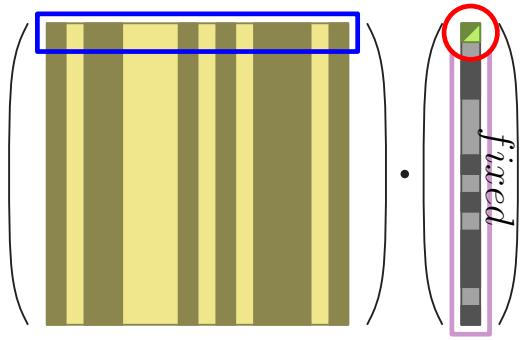
Randomisierte Algorithmen

Matrix-Matrix Multiplikation

Behauptung: Wenn $XY \neq Z$ dann $\mathbb{P}[XYr = Zr] \leq 0.5$

Definitionen:

- $A := XY$, $B := Z$



Annahme: $A \neq B$; Wann ist dann $Ar = Br$?

- $\exists i, j : A_{i,j} \neq B_{i,j}$
- Sei $a := A_i$ und $b := B_i$ A_i: i'te Zeile von A
  $\alpha := \sum_{k \neq j} a_k r_k$ und $\beta := \sum_{k \neq j} b_k r_k$
  $\Rightarrow ar = \alpha + a_j r_j$ und $br = \beta + b_j r_j$
  $\Rightarrow ar - br = (\alpha - \beta) + (a_j - b_j) r_j$
- Annahme: Werte für $r_{1 \ldots j-1, j+1 \ldots n}$ bereits festgelegt
  $\rightarrow$ noch 2 Möglichkeiten für r_j, Gleichheit bei maximal einer
  $\Rightarrow \mathbf{Pr}[ar - br = 0] = \mathbf{Pr}[r_j = \frac{\beta - \alpha}{a_j - b_j}] \leq \frac{1}{2}$

$\rightarrow$ Fehlerrate $p \leq 0.5$

Randomisierte Algorithmen

Matrix-Matrix Multiplikation

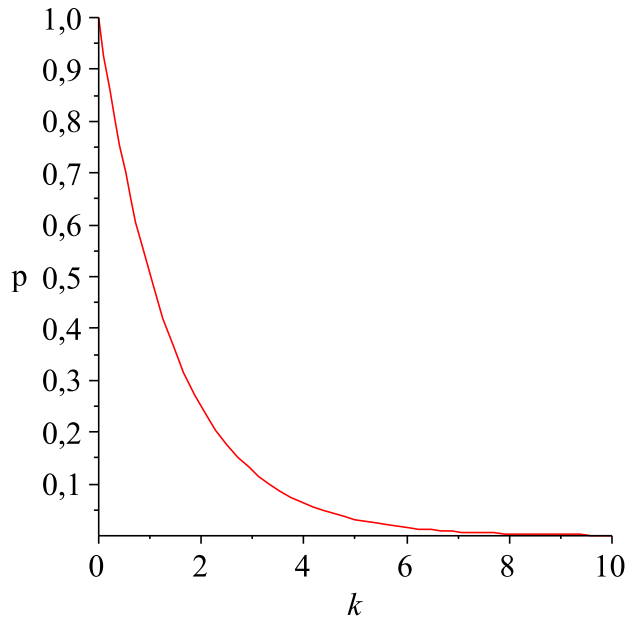
Beschleunigung durch *probability boosting*

nur bei $p \leq 0.5$ schnelle Konvergenz

- Wiederhole Test k mal mit unterschiedlicher Wahl von r
  - Ein Test liefert FALSCH
    $\rightarrow AB \neq C$, fertig
  - Alle Tests liefern KORREKT
    $\rightarrow$ *false positive* mit Wahrscheinlichkeit
    $\mathbb{P}[ABr = Cr] \leq 0.5^k$

$\rightarrow$ Laufzeit $\mathcal{O}(kn^2)$

linear längere Laufzeit bei exponentiell weniger Fehler



Coupon Collector

Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}$

Coupon Collector

Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}$

Coupon Collector

Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}$

Coupon Collector

Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}$

Coupon Collector

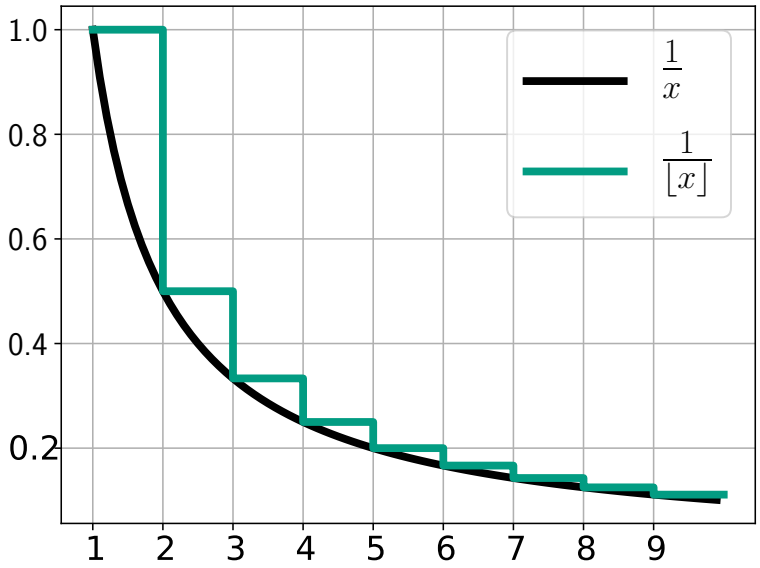
Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

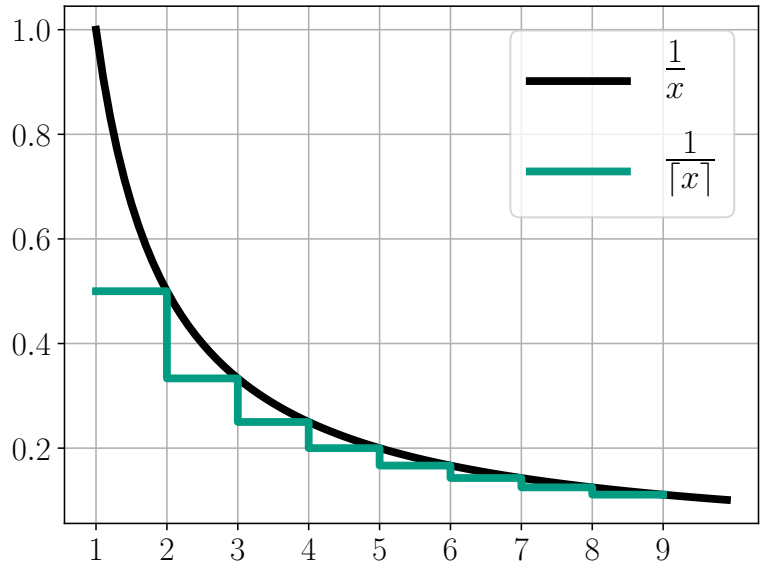
- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}$

Harmonische Zahlen

- $H_n = \sum_{i=1}^{n} \frac{1}{i}$

$$\ln n = \int_{x=1}^{n} \frac{1}{x} dx \leq \sum_{i=1}^{n} \frac{1}{i} \qquad \sum_{i=2}^{n} \frac{1}{i} \leq \int_{x=1}^{n} \frac{1}{x} dx = \ln n$$





- $\ln n \leq H_n \leq \ln n + 1$

Coupon Collector

Müslipackungen enthalten jeweils eine von n verschiedenen Sammelkarten. Wie viele Packungen muss ich kaufen um alle Karten beisammenzuhaben?

- X = # Packungen bis mind. eine von jeder Karte
- $X = \sum_{i=1}^{n} X_i$, mit X_i = # Packungen während ich $i-1$ Karten hatte
  - X_i sind geometrische Zufallsvariablen mit $p_i = 1 - \frac{i-1}{n}$
  - $\mathbb{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$
- $\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ Linearität des Erwartungswertes
  $= \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n \leq n \ln n + n$

Ende!

Feierabend!