

Übung 6 – Algorithmen II

Hans-Peter Lehmann, Daniel Seemaier – {hans-peter.lehmann, daniel.seemaier}@kit.edu
http://algo2.iti.kit.edu/AlgorithmenII_WS21.php

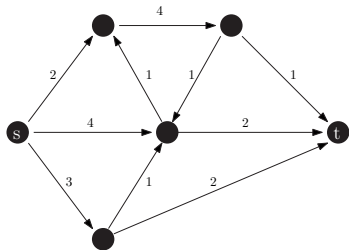
Institut für Theoretische Informatik - Algorithmik II

```
    result = current_weight;  
    return true;  
}  
  
for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){  
    const Edge & edge = graph.getEdge( eid );  
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )  
    if( edge.forward ){  
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )  
        weight new_weight = edge.weight + current_weight;  
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );  
        if( !priority_queue.isReached( edge.target ) ){  
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )  
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )  
            priority_queue.push( edge.target, new_weight );  
        } else {  
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){  
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )  
                priority_queue.decreaseKey( edge.target, new_weight );  
            }  
        }  
    }  
}
```

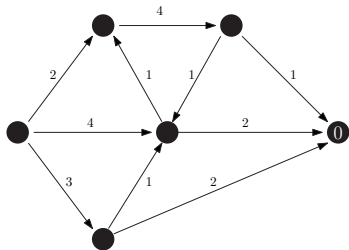
- Dinitz
- Potentialmethode
- Preflow-Push

Dinitz Algorithmus

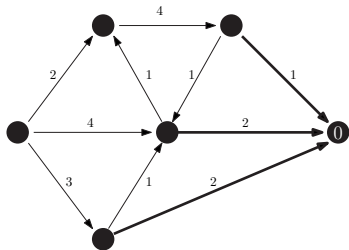
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
 auch: kürzeste Wege Netzwerk, Schichtgraph



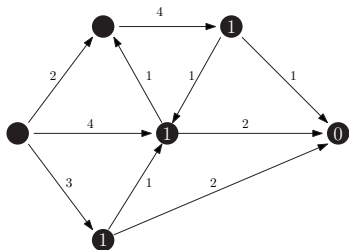
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
 auch: kürzeste Wege Netzwerk, Schichtgraph



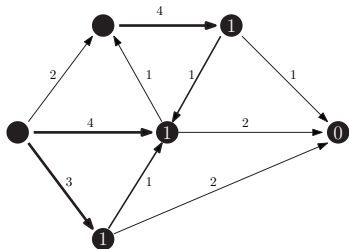
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



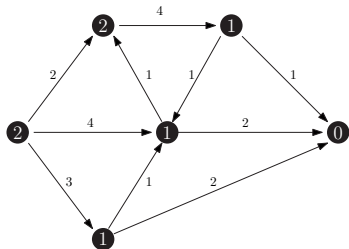
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



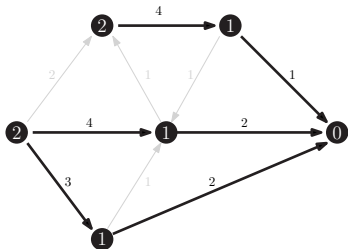
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



- 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



- Knotenmenge $V_S = V$
- $E' = \{e = (u, v) \in E \mid f(e) < c(e),$
 $d(u) = d(v) + 1\}$
 $E'^{rev} = \{e^{rev} = (v, u)^{rev} \mid f(v, u) > 0,$
 $d(u) = d(v) + 1\}$
 Kantenmenge $E_S = E' \cup E'^{rev}$
- betrachte Analogie zu Edmonds-Karp (kürzeste erhöhende Wege)



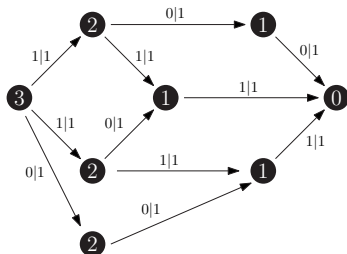
■ Kein weiterer Fluss möglich

“Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”

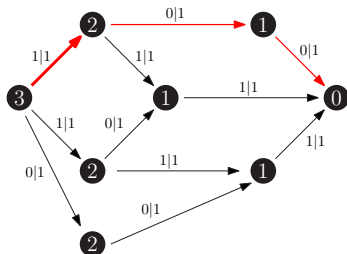
■ *Blocking flow als atomare Operation*

- Berechnung auf Schichtgraph
- Kein Residualgraph
- Kein Rückfluss möglich

■ i.A. nicht maximal auf Schichtgraph und Residualgraph



- 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 Residualgraph



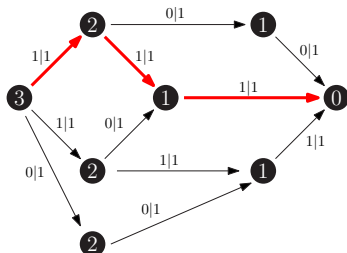
■ Kein weiterer Fluss möglich

“Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”

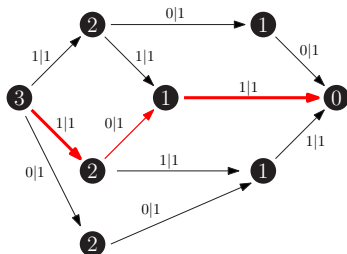
■ *Blocking flow als atomare Operation*

- Berechnung auf Schichtgraph
- Kein Residualgraph
- Kein Rückfluss möglich

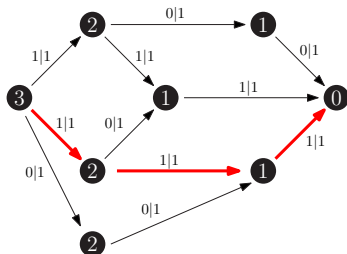
■ i.A. nicht maximal auf Schichtgraph und Residualgraph



- 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 Residualgraph



- 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 Residualgraph



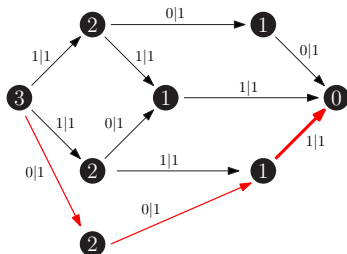
■ Kein weiterer Fluss möglich

“Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”

■ *Blocking flow als atomare Operation*

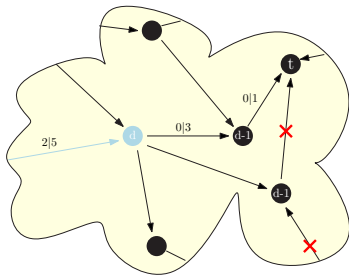
- Berechnung auf Schichtgraph
- Kein Residualgraph
- Kein Rückfluss möglich

■ i.A. nicht maximal auf Schichtgraph und Residualgraph

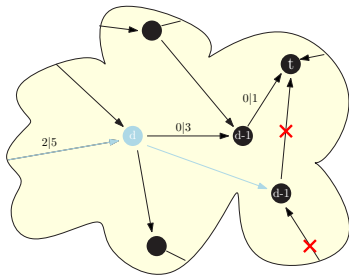


- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
 - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten

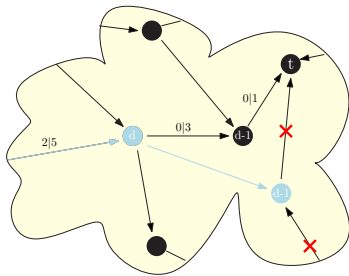
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



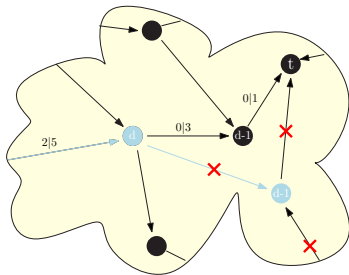
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



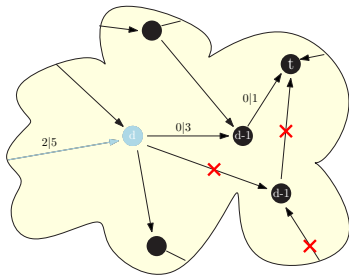
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



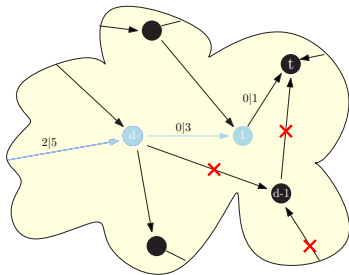
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



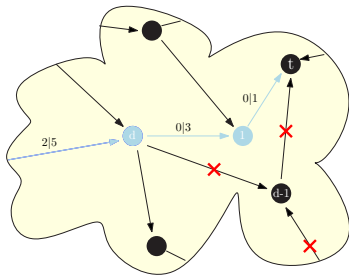
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



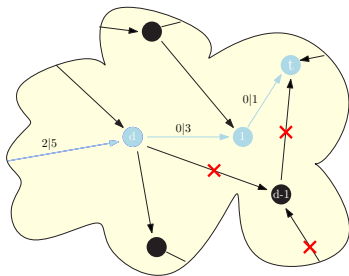
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



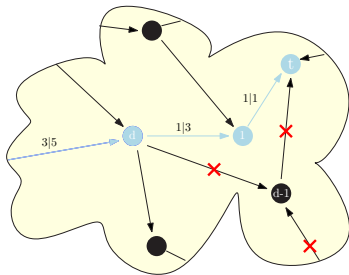
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



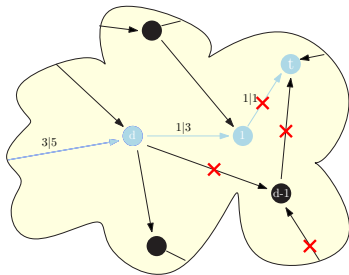
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



- *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 gesättigte Kanten



- *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



- $\#breakthrough \leq m$
 - Jedes breakthrough saturiert mindestens eine Kante
→ 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 $O(nm)$

- 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)

→ **Gesamtlaufzeit** $O(n^2m)$

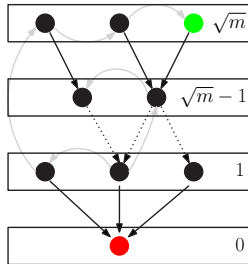
- Pro Phase
 - Rückwärtsgerichtete Breitensuche: **Laufzeit** $O(n + m)$
 - Blockierender Fluss: **Laufzeit** $O(nm)$
- Phase in $O(nm)$
- #Phasen $\leq n$ (Beweis, wie in Vorlesung, nicht hier)
(Lemma 1: Jede Phase erhöht Label von s um mindestens 1)
→ **Gesamtlaufzeit** $O(n^2m)$

Amortisierte Kosten

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

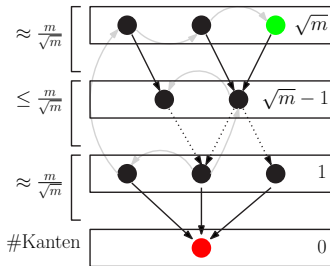
■ Nach $\sqrt{m}$ Phasen

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



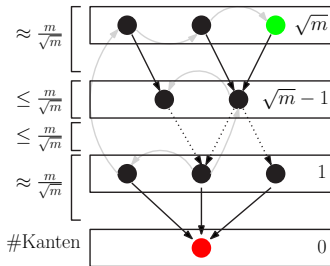
■ Nach $\sqrt{m}$ Phasen

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



■ Nach $\sqrt{m}$ Phasen

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



Potentialmethode

■ 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 $\mathcal{O}(n)$

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



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

■ 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 $\mathcal{O}(n)$

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

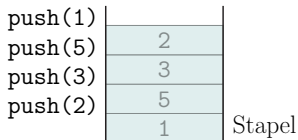


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ 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 $\mathcal{O}(n)$

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

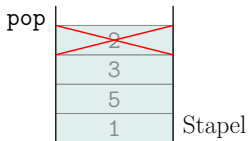


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ Datenstruktur Stapel mit `multiop`

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

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

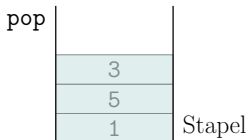


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ Datenstruktur Stapel mit `multiop`

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

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

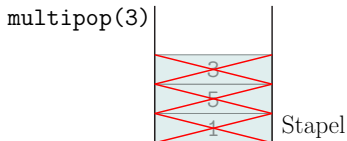


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ 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 $\mathcal{O}(n)$

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

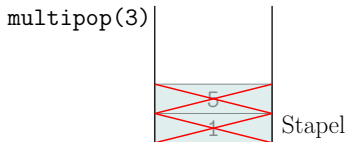


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ 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 $\mathcal{O}(n)$

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

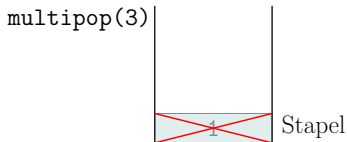


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ 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 $\mathcal{O}(n)$

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

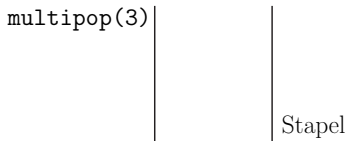


Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

■ 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 $\mathcal{O}(n)$

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



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

■ Sei $\Phi = \#_{\text{Elemente auf Stack}} \geq 0$

- $\text{push}(v)$: $\Delta\Phi = 1$ Erhöhung
- pop : $\Delta\Phi \in \{-1, 0\}$ Erniedrigung
- $\text{multi-pop}(k)$: $\Delta\Phi \in \{-k, \dots, 0\}$ Erniedrigung

- da $\#_{\text{push}} \leq n$ bei n Operationen
 - maximal n Erhöhungen
 - Erniedrigungen um maximal n möglich
- insgesamt Änderungen um $\leq 2n$

→ maximale Kosten $\mathcal{O}(n)$

Erhöhung oder Erniedrigung um 1 kostet $\mathcal{O}(1)$

→ Was ist mit pop / multi-pop 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 $\text{Level } d(v) = d(w) + 1$

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

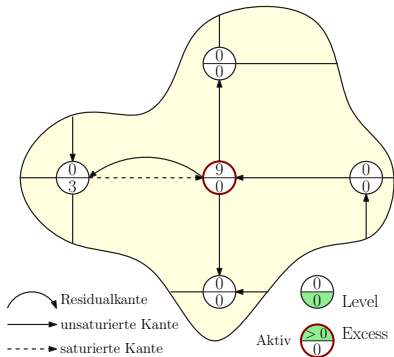
inflow > outflow

- **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)
- Voraussetzung: $d(u) = d(v) + 1$



preflow-push Algorithmus

Wiederholung

- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess

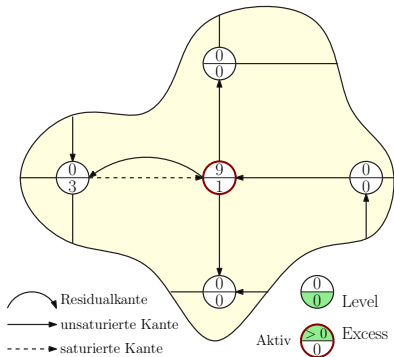
inflow > outflow

■ 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

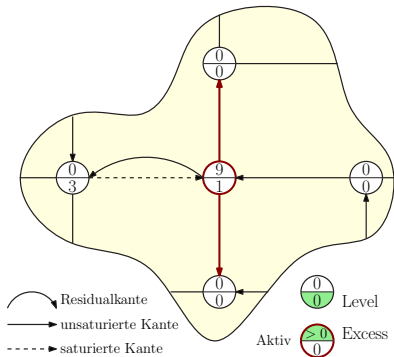
- schiebe Fluss entlang Kante (u, v)
- Voraussetzung: $d(u) = d(v) + 1$



preflow-push Algorithmus

Wiederholung

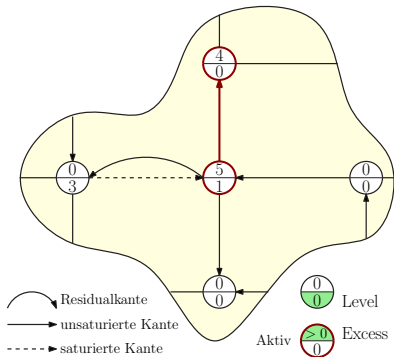
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
inflow > outflow
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

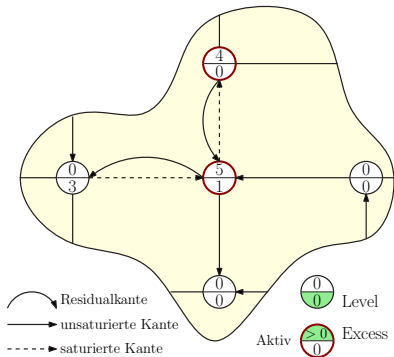
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
inflow > outflow
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



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)
 - Voraussetzung: $d(u) = d(v) + 1$



preflow-push Algorithmus

Wiederholung

- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess

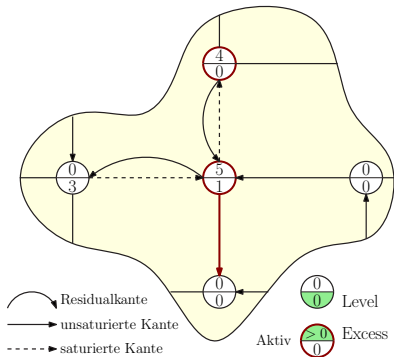
inflow > outflow

- 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**

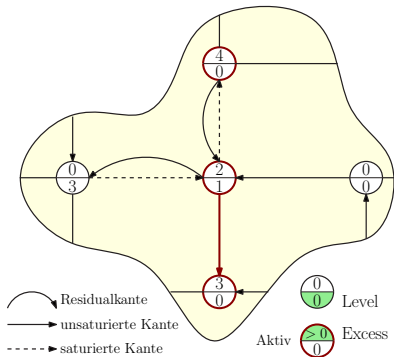
- schiebe Fluss entlang Kante (u, v)
- Voraussetzung: $d(u) = d(v) + 1$



preflow-push Algorithmus

Wiederholung

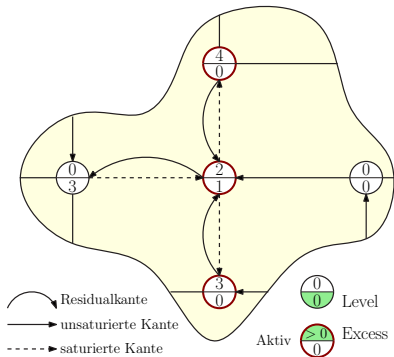
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
inflow > outflow
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



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)
 - Voraussetzung: $d(u) = d(v) + 1$



preflow-push Algorithmus

Wiederholung

- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess

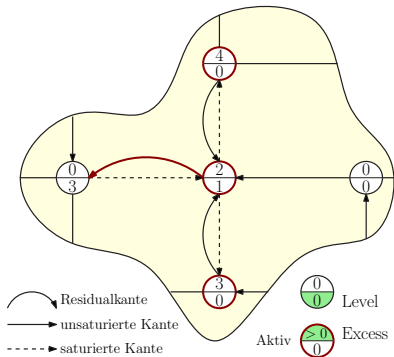
inflow > outflow

■ 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

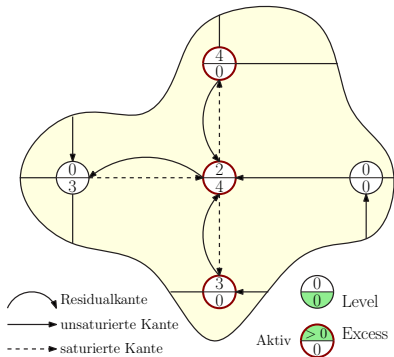
- schiebe Fluss entlang Kante (u, v)
- Voraussetzung: $d(u) = d(v) + 1$



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)
 - Voraussetzung: $d(u) = d(v) + 1$



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 $\text{Level } d(v) = d(w) + 1$

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 $\text{Level } d(v) = d(w) + 1$

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

WELCHEN?
WELCHE?
WIEVIEL?

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 $\text{Level } d(v) = d(w) + 1$

allgemeiner Ablauf

1. wähle aktiven Knoten v
2. falls gültige Kante (v, w) existiert: push
 - schiebe Fluss entlang (v, w)
 $f_{(v,w)} = f_{(v,w)} + \min\{c_{(v,w)}^f, \text{excess}(v)\}$
3. ansonsten: relabel
 - erhöhe Level von v
 $d(v) = d(v) + 1$

WELCHEN?
WELCHE?
WIEVIEL?

WIEVIEL?

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*

→ nur **Heuristiken**, aber in Praxis deutliche Beschleunigung!

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

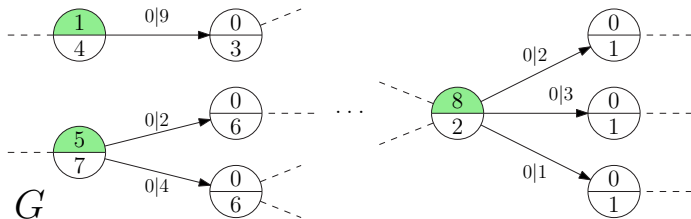
FIFO preflow-push Algorithmus

Beweis Laufzeit

■ Unterteile Ablauf in **Phasen**:

- alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
 - pro Phase baut jeder Knoten max. 1x allen *excess* ab
 - pro Phase macht jeder Knoten max. 1x einen *non-saturating* push

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



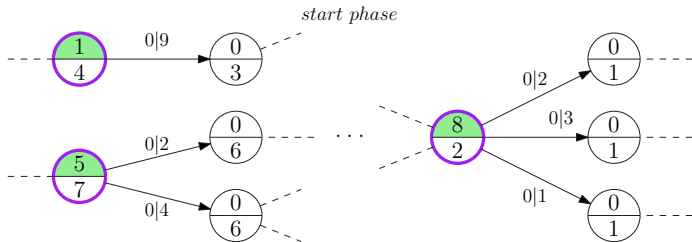
FIFO preflow-push Algorithmus

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

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



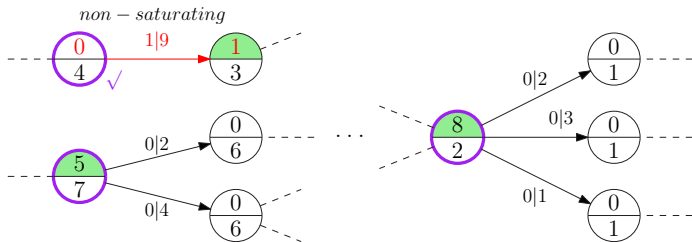
FIFO preflow-push Algorithmus

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

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



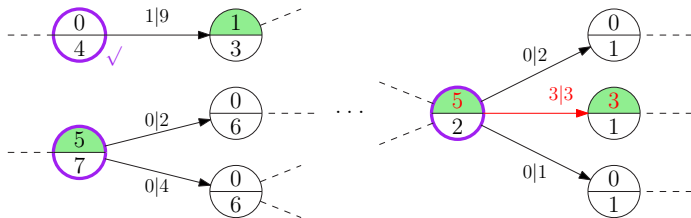
FIFO preflow-push Algorithmus

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

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



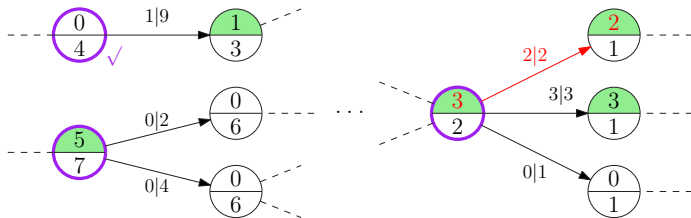
FIFO preflow-push Algorithmus

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

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



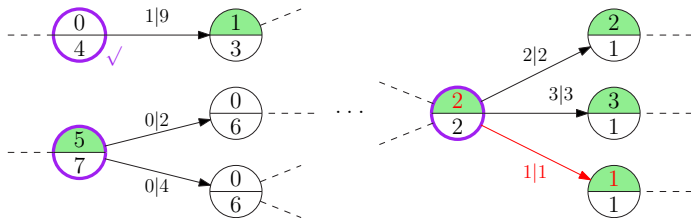
FIFO preflow-push Algorithmus

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

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



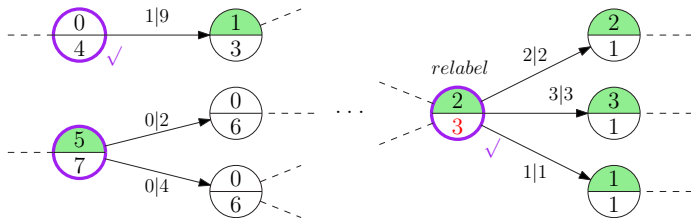
FIFO preflow-push Algorithmus

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

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



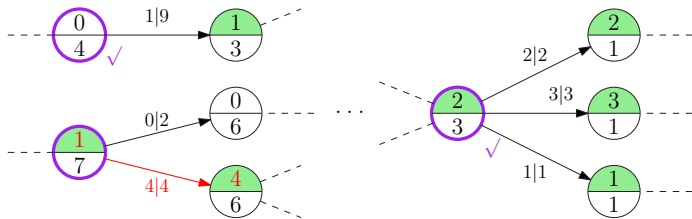
FIFO preflow-push Algorithmus

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

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



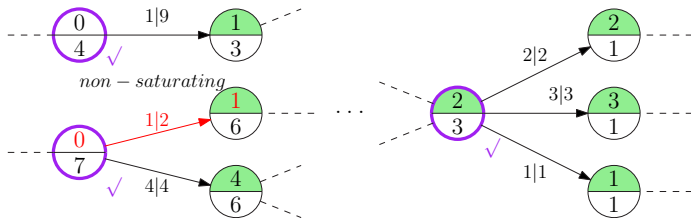
FIFO preflow-push Algorithmus

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

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



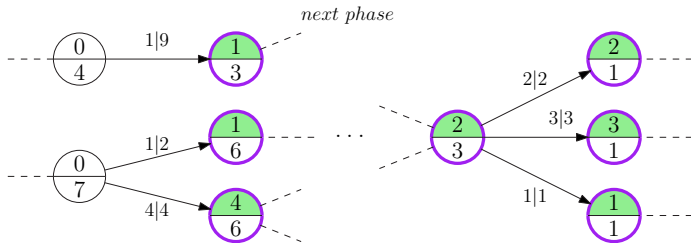
FIFO preflow-push Algorithmus

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

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



FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$

Potential, das 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

→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- 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, das 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

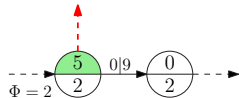
→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- 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, das 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

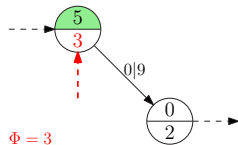
→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- 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, das gleich dem höchsten aktiven Level ist

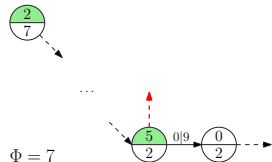
- **Phase mit relabel:** $\Delta\Phi \leq 1$ Erhöhung
Phase ohne relabel: $\Delta\Phi \leq -1$ Erniedrigung

- da $\#_{\text{relabel}} \leq 2n^2$ siehe Vorlesung
→ maximal $2n^2$ Erhöhungen
→ 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, das 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

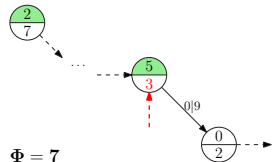
→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- 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, das 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

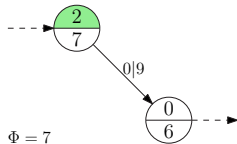
→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- 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, das gleich dem höchsten aktiven Level ist

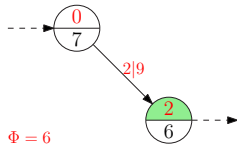
- Phase mit relabel: $\Delta\Phi \leq 1$ Erhöhung
Phase ohne relabel: $\Delta\Phi \leq -1$ Erniedrigung

- da $\#_{\text{relabel}} \leq 2n^2$ siehe Vorlesung
→ maximal $2n^2$ Erhöhungen
→ 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, das gleich dem höchsten aktiven Level ist

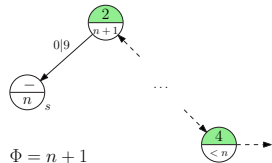
- Phase mit relabel: $\Delta\Phi \leq 1$ Erhöhung
Phase ohne relabel: $\Delta\Phi \leq -1$ Erniedrigung

- da $\#_{\text{relabel}} \leq 2n^2$ siehe Vorlesung
→ maximal $2n^2$ Erhöhungen
→ 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, das gleich dem höchsten aktiven Level ist

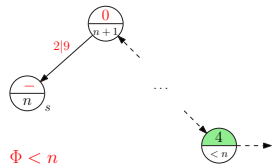
- Phase mit relabel: $\Delta\Phi \leq 1$ Erhöhung
Phase ohne relabel: $\Delta\Phi \leq -1$ Erniedrigung

- da $\#_{\text{relabel}} \leq 2n^2$ siehe Vorlesung
→ maximal $2n^2$ Erhöhungen
→ 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, das 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

→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

- restlicher Beweis wie bei *generic preflow-push*

FIFO preflow-push Algorithmus

Laufzeit

Lemmas

- $T_{\text{push op}} = T_{\text{relabel op}} = T_{\text{node selection}} = \mathcal{O}(1)$

- $\#\text{relabels} \leq 2n^2$

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

- $\#\text{non-saturating} \in \mathcal{O}(n^2m)$

Lemma 9

- $T_{\text{generic preflow-push}} = T_{\text{init}} + T_{\text{pushes}} + T_{\text{relabels}} \in \mathcal{O}(n^2m)$

- $T_{\text{init}} = n + m$

- $T_{\text{pushes}} = \#\text{pushes} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\text{push op}})$

- $T_{\text{relabels}} = \#\text{relabels} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\text{relabel op}})$

FIFO preflow-push Algorithmus

Laufzeit

Lemmas

- $T_{\text{push op}} = T_{\text{relabel op}} = T_{\text{node selection}} = \mathcal{O}(1)$

- $\#\text{relabels} \leq 2n^2$

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

- $T_{\text{FIFO preflow-push}} = T_{\text{init}} + T_{\text{pushes}} + T_{\text{relabels}} \in \mathcal{O}(n^3)$

- $T_{\text{init}} = n + m$

- $T_{\text{pushes}} = \#\text{pushes} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\text{push op}})$

- $T_{\text{relabels}} = \#\text{relabels} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\text{relabel op}})$

Relabeling

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

Knotenauswahl

- *two-phase approach*

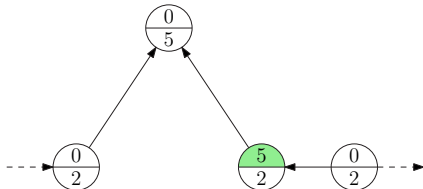
■ aggressive local relabeling

- 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



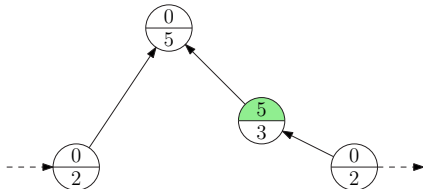
■ aggressive local relabeling

- 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



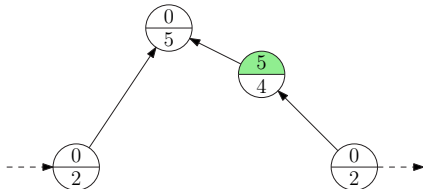
■ aggressive local relabeling

- 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

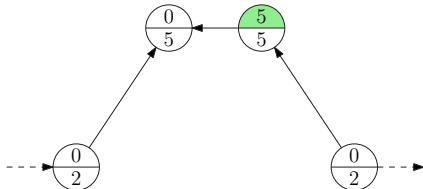
■ aggressive local relabeling

- 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

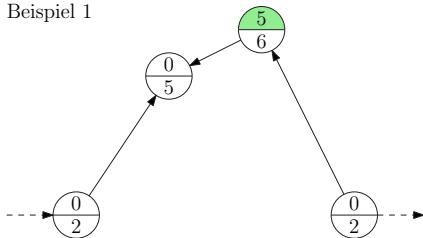
■ aggressive local relabeling

- 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



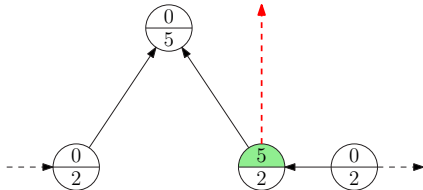
■ aggressive local relabeling

- 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



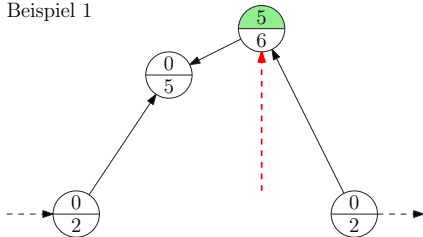
■ aggressive local relabeling

- 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



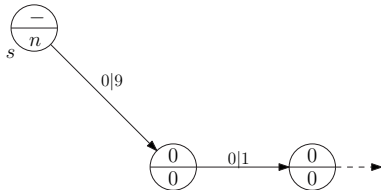
■ aggressive local relabeling

- 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



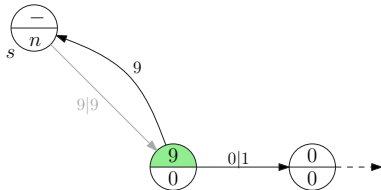
■ aggressive local relabeling

- 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



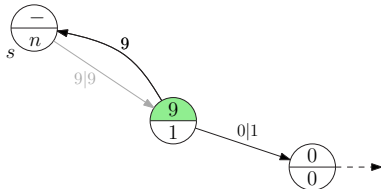
■ aggressive local relabeling

- 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



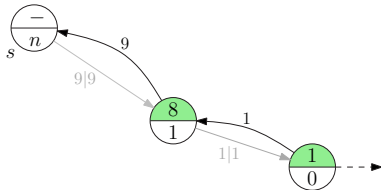
■ aggressive local relabeling

- 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



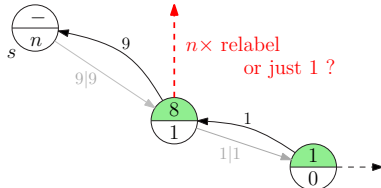
■ aggressive local relabeling

- 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

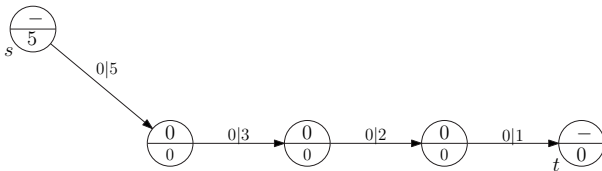
■ *global relabeling*

- setze Levels auf

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

- 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

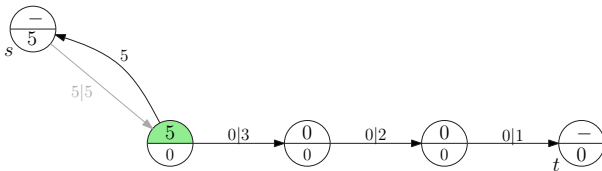
■ global relabeling

- setze Levels auf

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

- 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

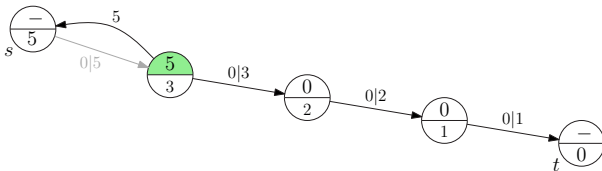
■ global relabeling

- setze Levels auf

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

- 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

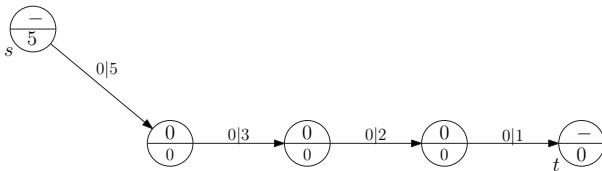
■ *global relabeling*

- setze Levels auf

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

- 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

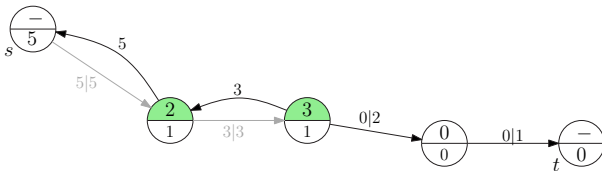
■ global relabeling

- setze Levels auf

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

- 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

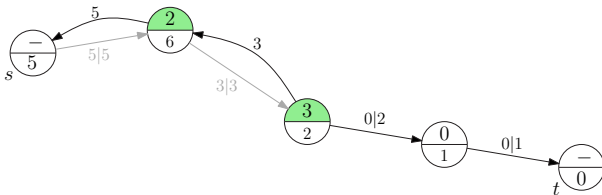
■ global relabeling

- setze Levels auf

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

- 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

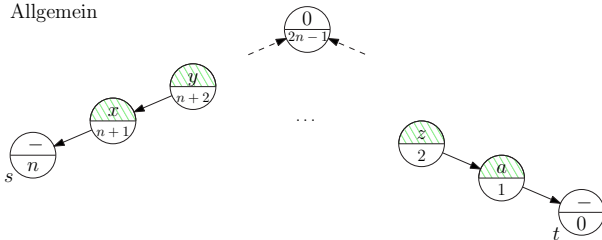
■ global relabeling

- setze Levels auf

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

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

Allgemein



preflow-push Algorithmus

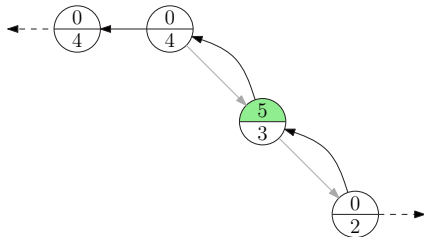
Heuristiken

■ 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

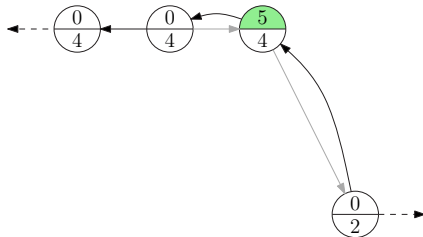


■ *gap heuristic*

- 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

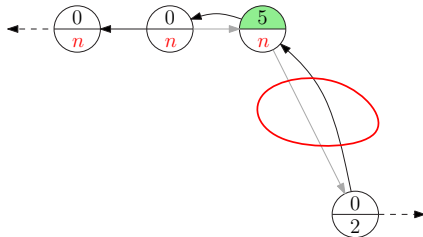
Heuristiken

■ 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



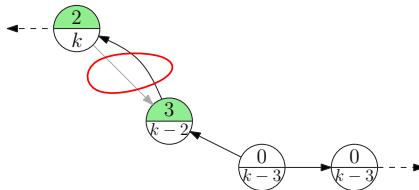
■ gap heuristic

- 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

Allgemein

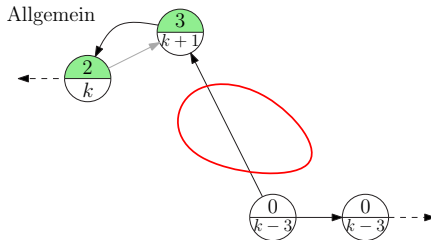


■ gap heuristic

- 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

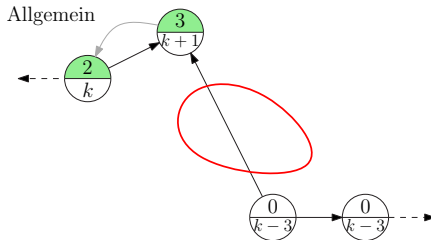
Heuristiken

■ 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

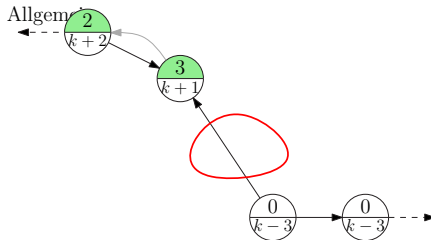


■ gap heuristic

- 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

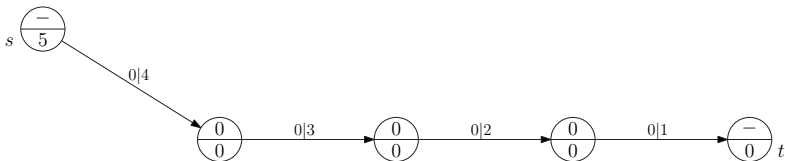


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

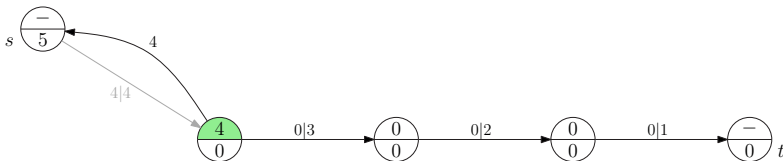


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

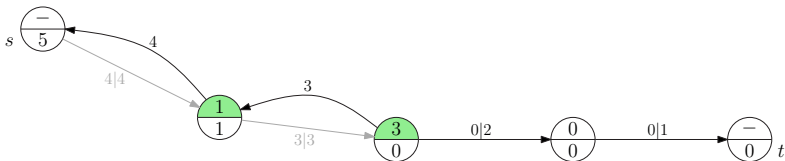


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

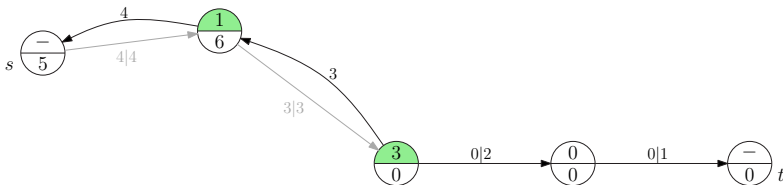


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

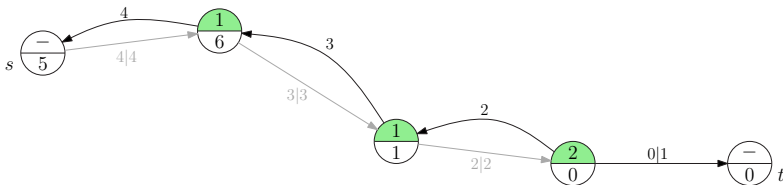


preflow-push Algorithmus

Heuristiken

■ two-phase approach

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

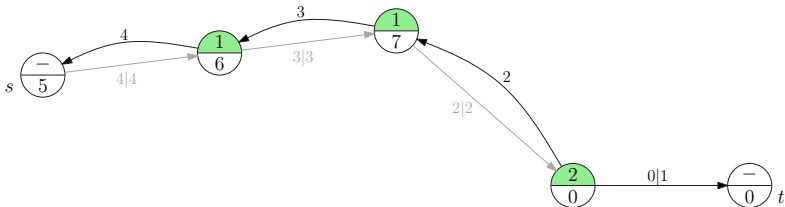


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

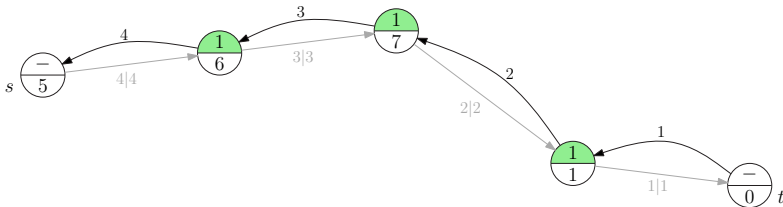


preflow-push Algorithmus

Heuristiken

■ two-phase approach

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

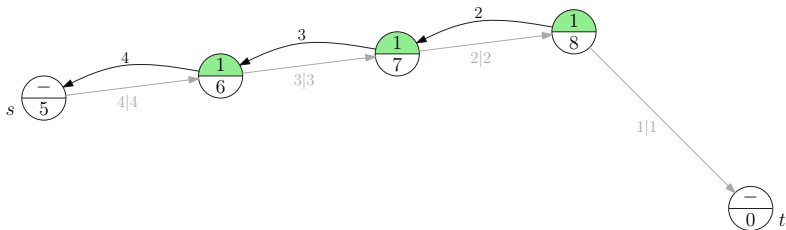


preflow-push Algorithmus

Heuristiken

■ two-phase approach

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

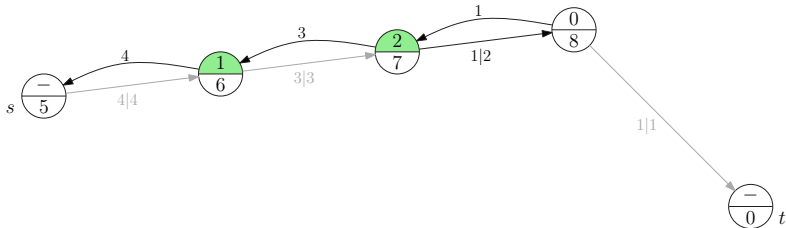


preflow-push Algorithmus

Heuristiken

■ *two-phase approach*

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

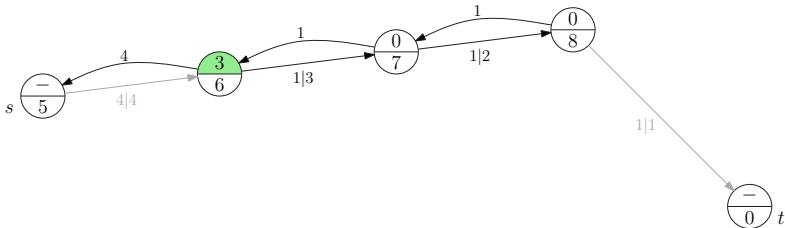


preflow-push Algorithmus

Heuristiken

■ two-phase approach

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

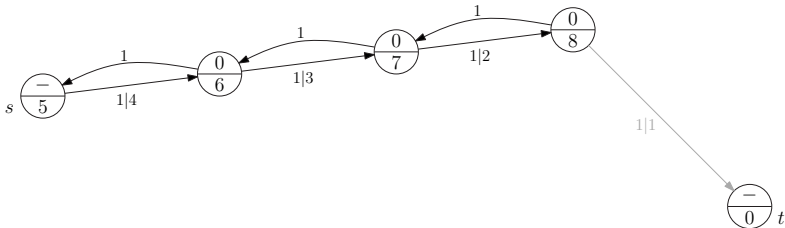


preflow-push Algorithmus

Heuristiken

■ two-phase approach

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



Ende!



Feierabend!