

Übung 13 – Algorithmen II

Yaroslav Akhremtsev, Demian Hespe – yaroslav.akhremtsev@kit.edu, hespe@kit.edu

Mit Folien von Michael Axtmann

http://algo2.iti.kit.edu/AlgorithmenII_WS17.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 );
            }
        }
    }
}
```

***preflow-push* Algorithmus**

- Überblick
- *FIFO preflow-push*
- Heuristiken

Matching

Speichermodell

- Parallel Disk Model
- Speicherlatenzen
- Blockgrößen

I/O-effizientes Design

- Basistechniken
- *externes* Sortieren

preflow-push Algorithmus

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

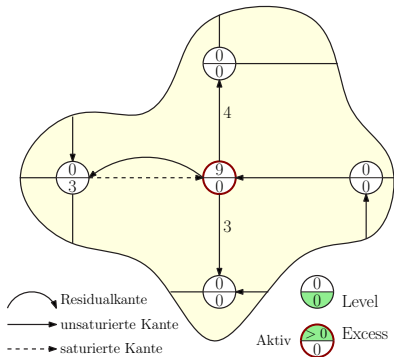
allgemeiner Ablauf

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

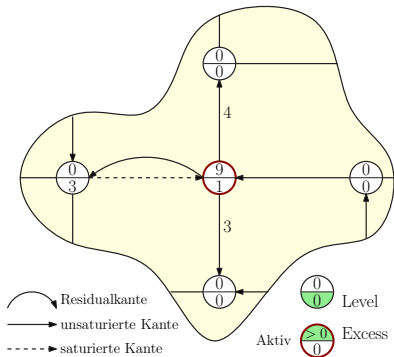
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- **relabel**
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

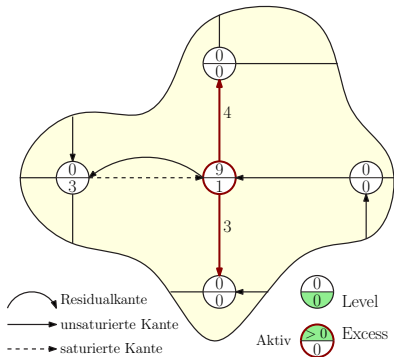
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- **relabel**
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

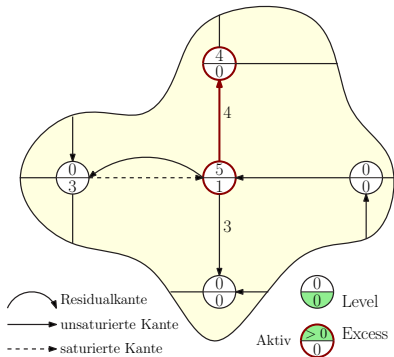
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

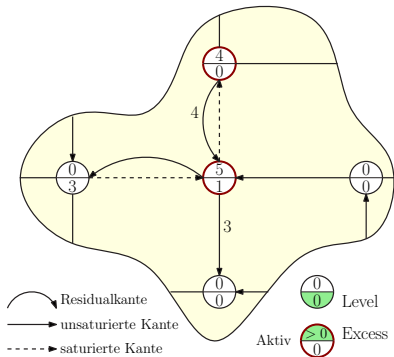
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

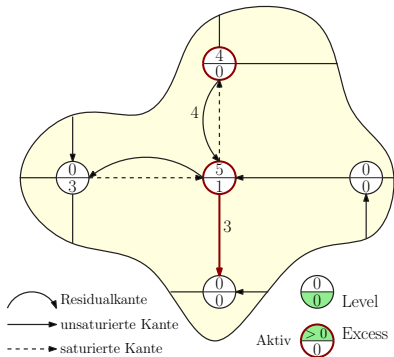
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

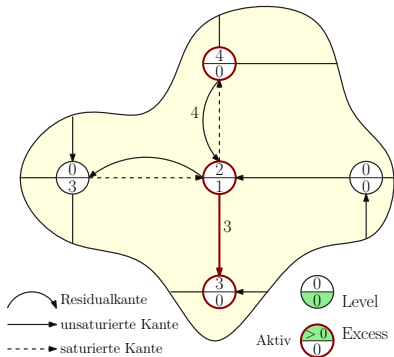
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

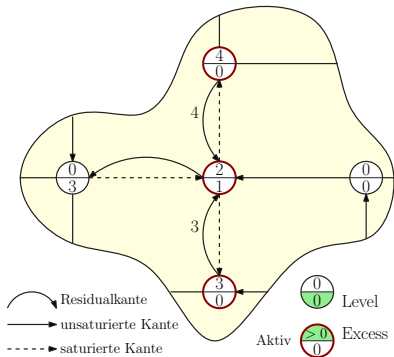
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

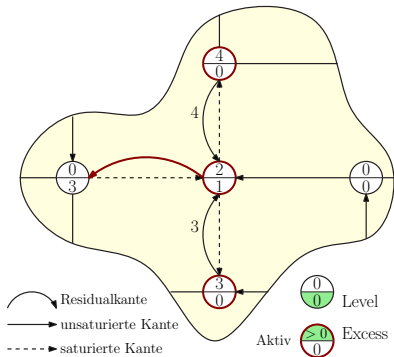
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- relabel
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



preflow-push Algorithmus

Wiederholung

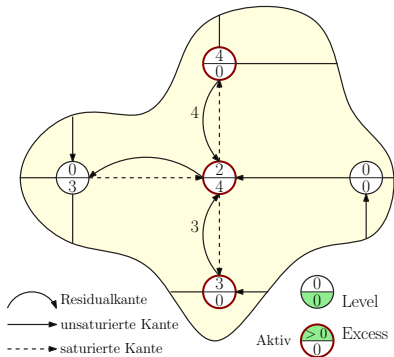
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
(inflow > outflow)
- **relabel**
 - erhöht Level eines Knoten
 - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
 - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
 - schiebe Fluss entlang Kante (u, v)
 - Voraussetzung: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



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 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

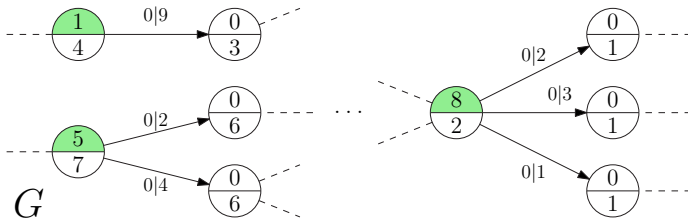
FIFO preflow-push Algorithmus

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

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



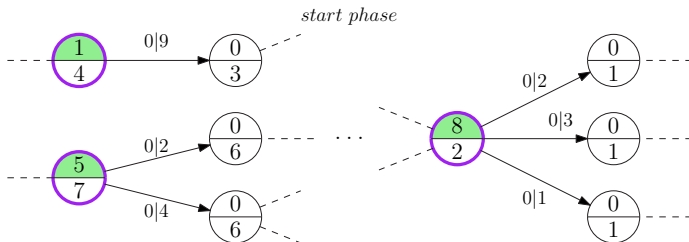
FIFO preflow-push Algorithmus

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

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



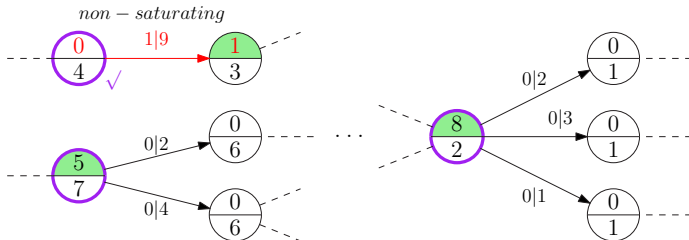
FIFO preflow-push Algorithmus

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

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



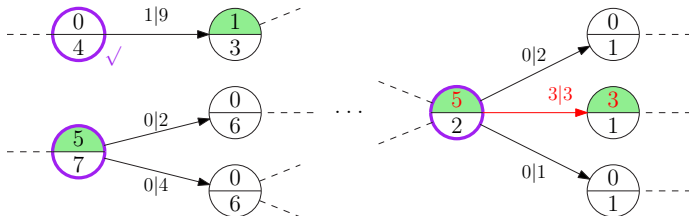
FIFO preflow-push Algorithmus

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

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



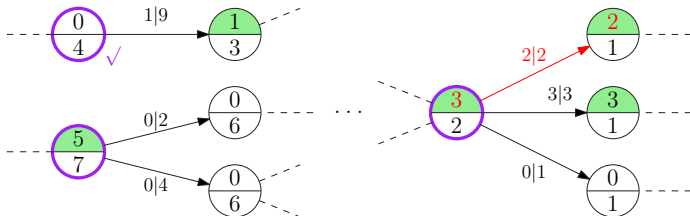
FIFO preflow-push Algorithmus

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

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



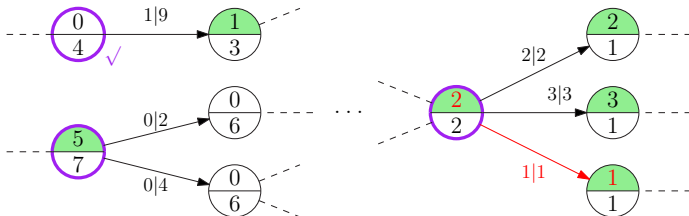
FIFO preflow-push Algorithmus

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

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



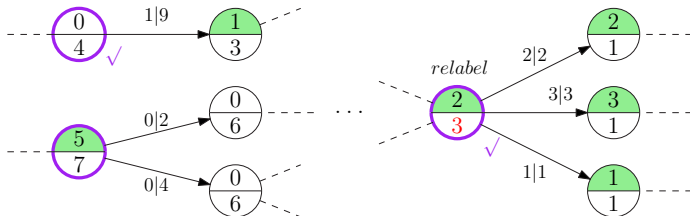
FIFO preflow-push Algorithmus

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

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



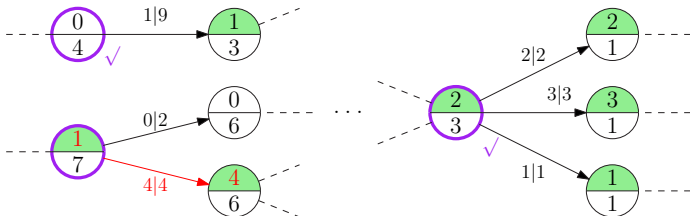
FIFO preflow-push Algorithmus

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

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



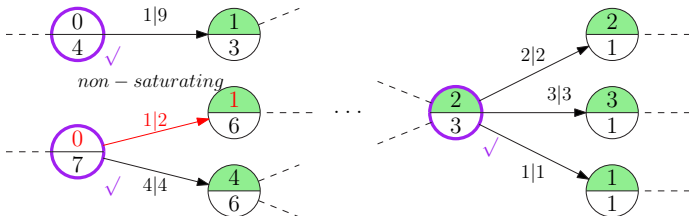
FIFO preflow-push Algorithmus

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

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



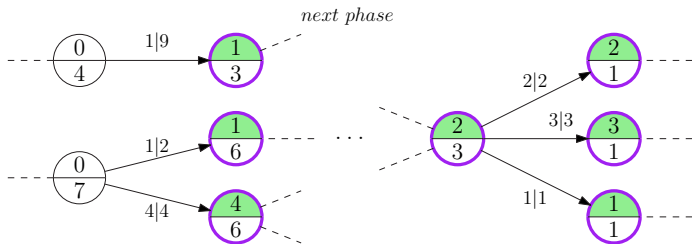
FIFO preflow-push Algorithmus

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, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

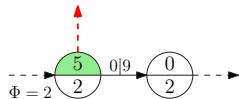
→ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*

FIFO preflow-push Algorithmus

Beweis Laufzeit

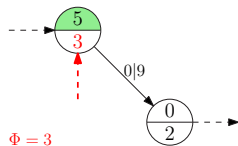
- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
(Potential, dass gleich dem höchsten aktiven Level ist)
 - Phase mit relabel: $\Delta\Phi \leq 1$ (Erhöhung)
Phase ohne relabel: $\Delta\Phi \leq -1$ (Erniedrigung)
 - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
 - maximal $2n^2$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

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

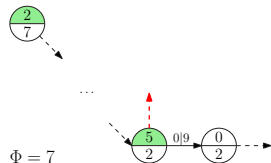
→ maximal $2n^2$ Erhöhungen

→ maximal $2n^2$ Erniedrigungen

→ $\#_{\text{phases}} \leq 4n^2$

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

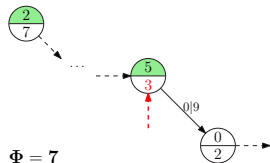
- restlicher Beweis wie bei *generic preflow-push*



FIFO preflow-push Algorithmus

Beweis Laufzeit

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

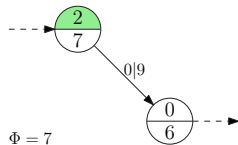
(Potential, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
 - maximal $2n^2$ Erhöhungen
 - maximal $2n^2$ Erniedrigungen
- $\#_{\text{phases}} \leq 4n^2$

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

- restlicher Beweis wie bei *generic preflow-push*



FIFO preflow-push Algorithmus

Beweis Laufzeit

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

(Potential, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

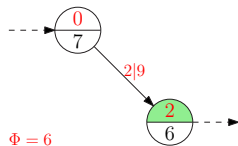
→ maximal $2n^2$ Erhöhungen

→ 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, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

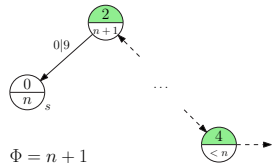
→ maximal $2n^2$ Erhöhungen

→ 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, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

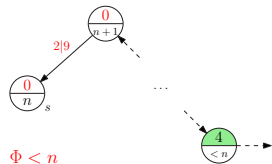
→ maximal $2n^2$ Erhöhungen

→ 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, dass gleich dem höchsten aktiven Level ist)

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

- da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)

→ 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*

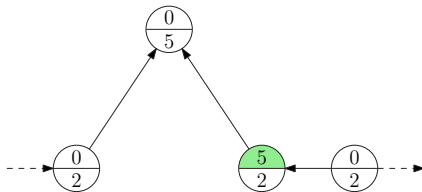
■ *aggressive local relabeling*

- 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



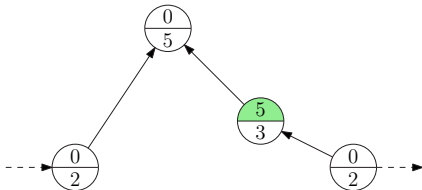
■ aggressive local relabeling

- 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



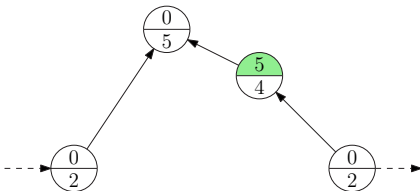
■ *aggressive local relabeling*

- 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

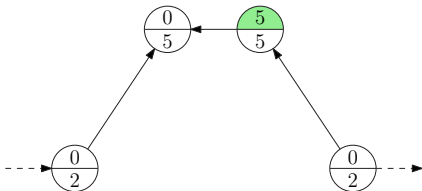
■ aggressive local relabeling

- 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

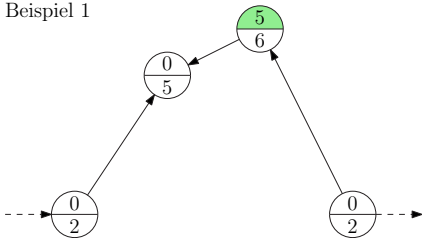
■ *aggressive local relabeling*

- 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



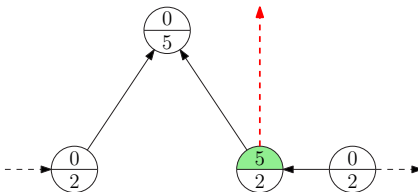
■ aggressive local relabeling

- 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



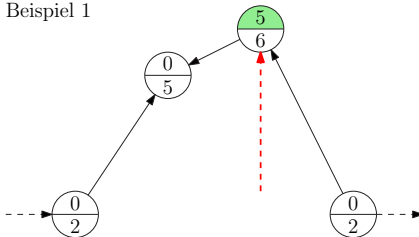
■ aggressive local relabeling

- 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



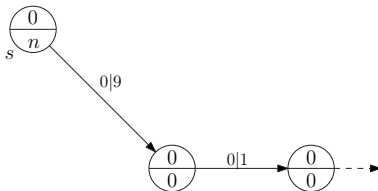
■ aggressive local relabeling

- 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



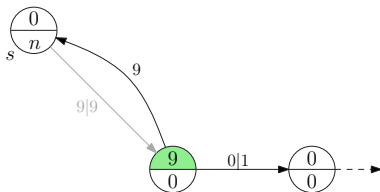
■ aggressive local relabeling

- 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



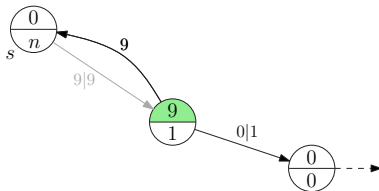
■ aggressive local relabeling

- 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



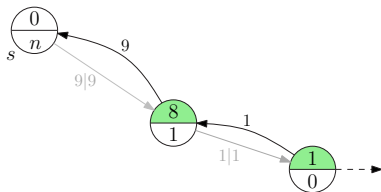
■ aggressive local relabeling

- 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



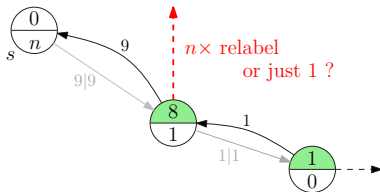
■ aggressive local relabeling

- 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

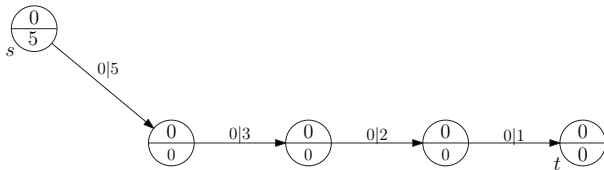
■ *global relabeling*

- 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

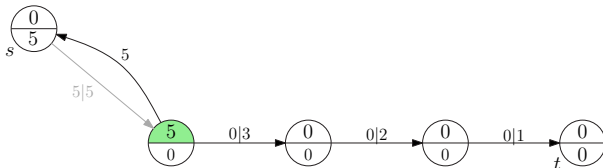
■ *global relabeling*

- 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

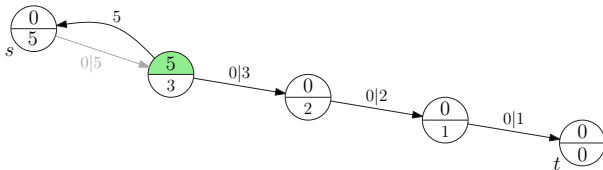
■ *global relabeling*

- 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

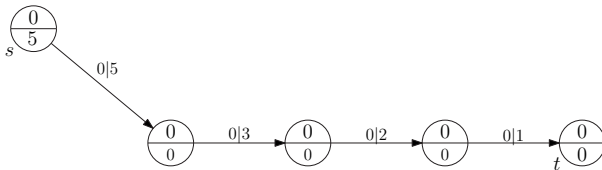
■ *global relabeling*

- 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

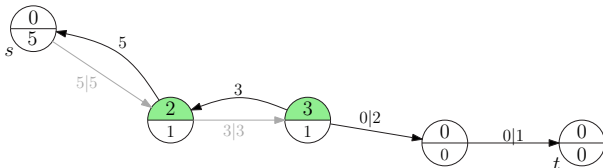
■ *global relabeling*

- 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

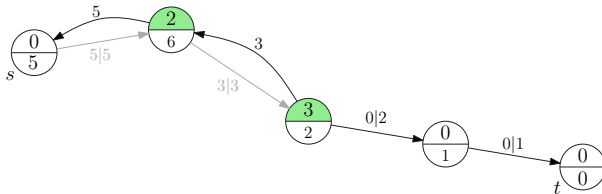
■ *global relabeling*

- 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

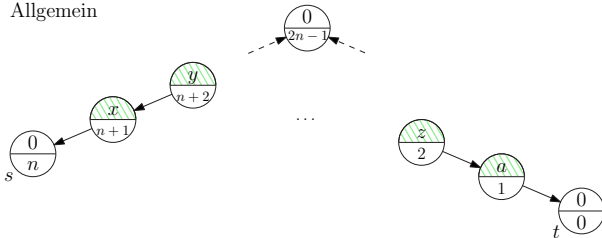
■ *global relabeling*

- 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

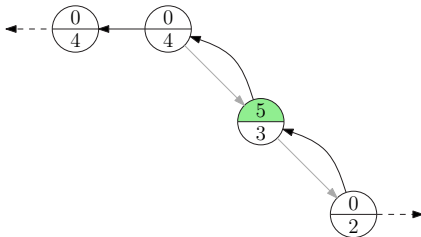


■ *gap heuristic*

- 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)

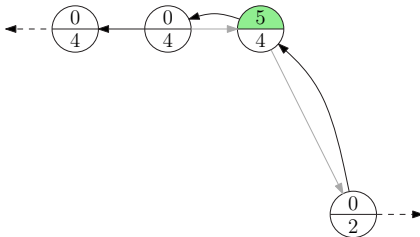


■ *gap heuristic*

- 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)

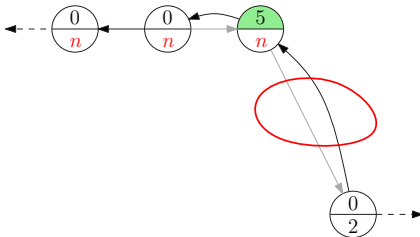


■ *gap heuristic*

- 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)



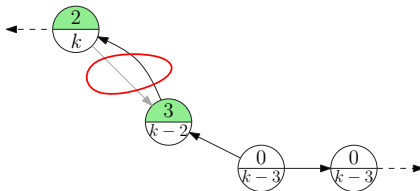
■ *gap heuristic*

- 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

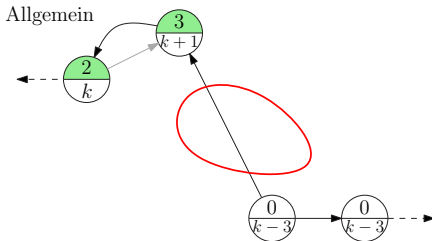


■ *gap heuristic*

- 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)

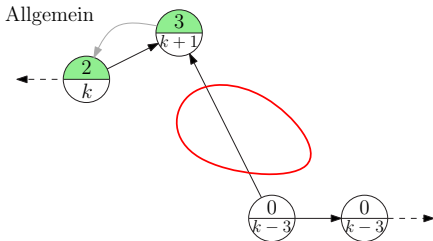


■ *gap heuristic*

- 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)

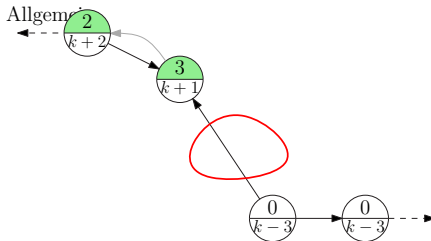


■ *gap heuristic*

- 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)

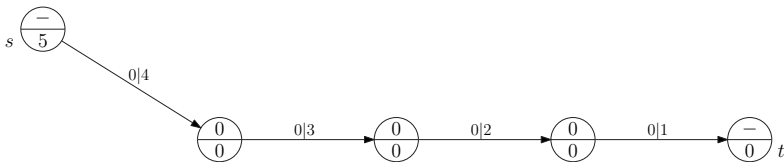


preflow-push Algorithmus

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

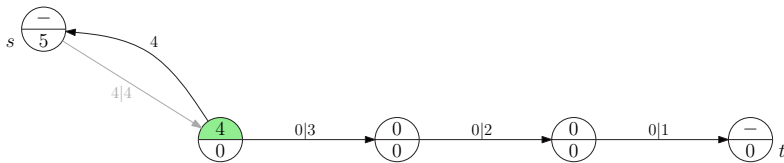


preflow-push Algorithmus

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

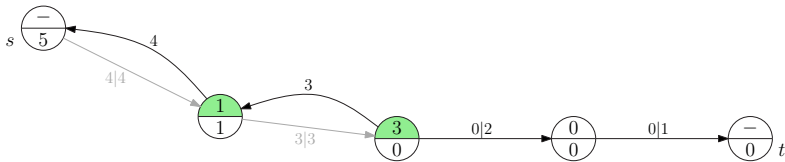


preflow-push Algorithmus

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

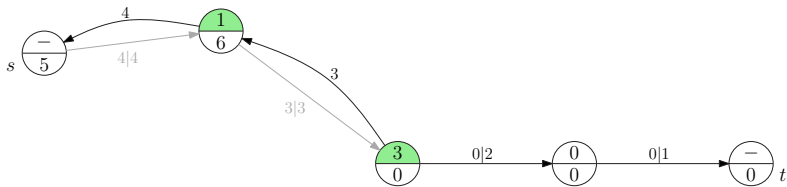


preflow-push Algorithmus

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

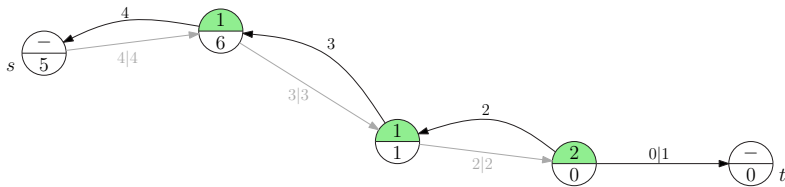


preflow-push Algorithmus

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

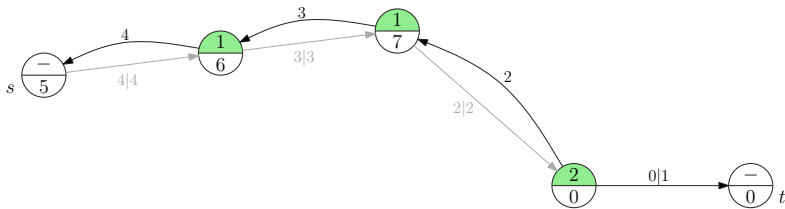


preflow-push Algorithmus

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

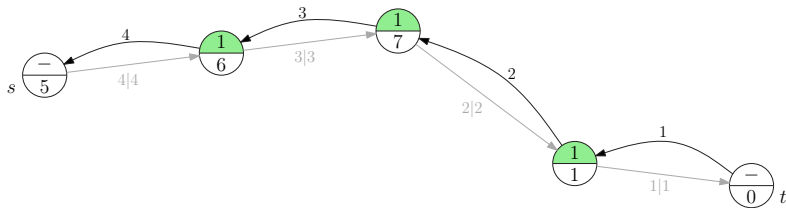


preflow-push Algorithmus

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

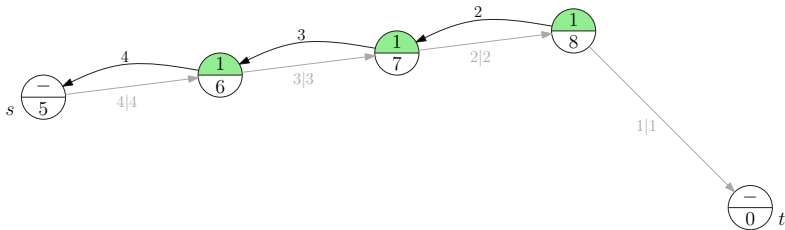


preflow-push Algorithmus

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

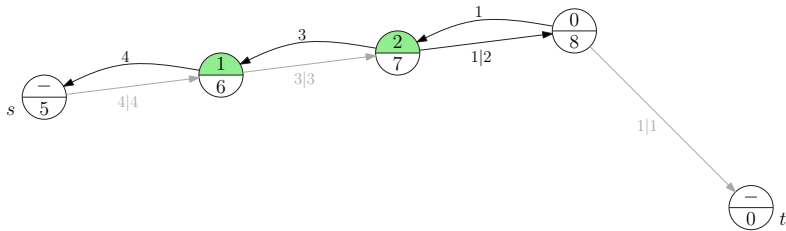


preflow-push Algorithmus

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

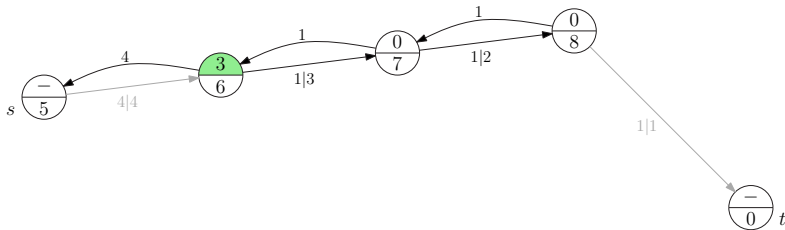


preflow-push Algorithmus

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

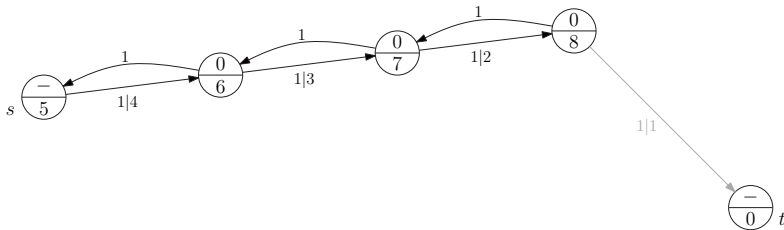


preflow-push Algorithmus

Heuristiken

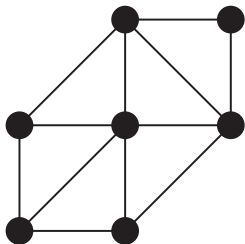
■ *two-phase approach*

- 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

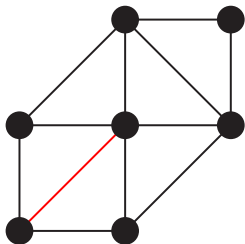


Matching

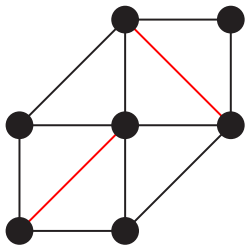
- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching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



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

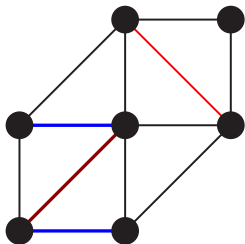


- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching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

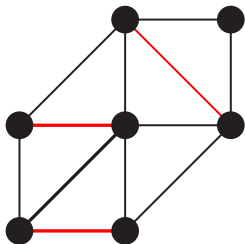


**maximal aber nicht
maximum**

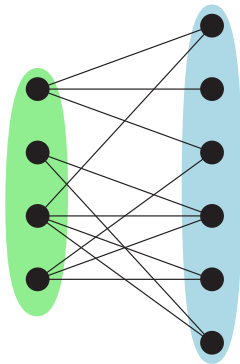
- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching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



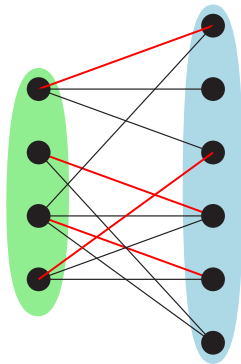
- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching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



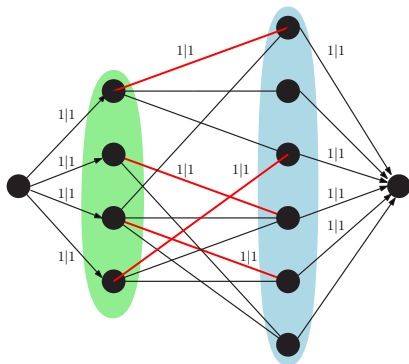
- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Kapazitäten 1
- Fluss zwischen den Teilmengen entspricht gematchten Kanten



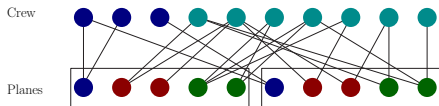
- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Kapazitäten 1
- Fluss zwischen den Teilmengen entspricht gematchten Kanten



- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Kapazitäten 1
- Fluss zwischen den Teilmengen entspricht gematchten Kanten

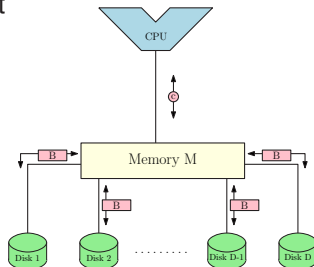


- Anwendungen in sämtlichen Zuweisungsproblemen
- ... Jobs auf Maschinen
- ... Crewscheduling

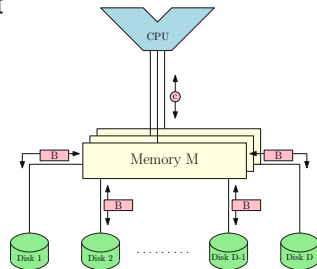


Speichermodell

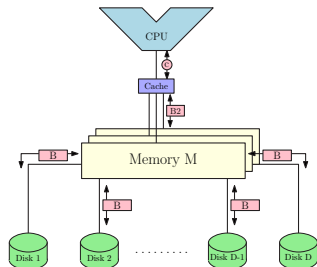
- Vitter und Shriver:
 - **Parallel Disk Model** (PDM)
- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren → Datenlokalität
- Muster wiederholt sich in Speicherhierarchie immer wieder



- Vitter und Shriver:
 - **Parallel Disk Model** (PDM)
- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren → Datenlokalität
- Muster wiederholt sich in Speicherhierarchie immer wieder

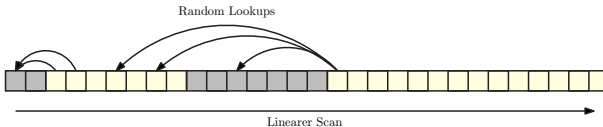


- Vitter und Shriver:
 - **Parallel Disk Model** (PDM)
- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren → Datenlokalität
- Muster wiederholt sich in Speicherhierarchie immer wieder

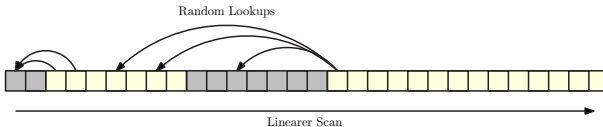


- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
(z.B. Europa)
 - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
 - Breitensuche (untere Schranke?): ≈ 2 Sek.

- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
(z.B. Europa)
 - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
 - Breitensuche (untere Schranke?): ≈ 2 Sek.
 - PHAST (linearer Scan): < 0.2 Sek.



- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
(z.B. Europa)
 - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
 - Breitensuche (untere Schranke?): ≈ 2 Sek.
 - PHAST (linearer Scan): < 0.2 Sek.



- **Strukturierter Zugriff** als wichtiges Designprinzip

Einflussfaktoren

- T_{seek} : Positionierungszeit
- W_{max} : maximale Bandbreite

→ Lesedauer: $T = T_{seek} + B/W_{max}$

Optimale Blockgröße

- Beispiel: RAM
 - $T_{seek} = 30$ ns (Zykluszeit)
 - $W_{max} \approx 25$ GB/s
 - Anzahl Blöcke pro Zeile: $R = 128$
 - Ziel: 95% Auslastung der Bandbreite bei sequenziellem Lesen
 - $W = \frac{B}{T} = B / \left(\frac{1}{128} T_{seek} + B/W_{max} \right) \stackrel{!}{=} 0.95 \cdot W_{max}$
 - $B = 0.15 \cdot W_{max} \cdot T_{seek} = 111$ Byte!

I/O-effizientes Design

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} \left(1 + \lceil \log_{M/B} \frac{n}{M} \rceil\right)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} \left(1 + \lceil \log_{M/B} \frac{n}{M} \rceil\right)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} \left(1 + \lceil \log_{M/B} \frac{n}{M} \rceil\right)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} (1 + \lceil \log_{M/B} \frac{n}{M} \rceil)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} (1 + \lceil \log_{M/B} \frac{n}{M} \rceil)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

■ Zugriffsmuster

- Random Access erwartet $\mathcal{O}(n)$ I/Os
- Linearer Scan $\mathcal{O}(n/B)$ I/Os

■ Sortieren

- $\mathcal{O}\left(\frac{2n}{B} (1 + \lceil \log_{M/B} \frac{n}{M} \rceil)\right)$ I/Os
- lokale Kriterien
- oft vorbereitend für linearen Scan

■ Prioritätswarteschlangen

- nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

■ Stack / Queue

- Zugriffsmuster
 - Random Access erwartet $\mathcal{O}(n)$ I/Os
 - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Sortieren
 - $\mathcal{O}\left(\frac{2n}{B} (1 + \lceil \log_{M/B} \frac{n}{M} \rceil)\right)$ I/Os
 - lokale Kriterien
 - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
 - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“
- Stack / Queue

■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- **alle** Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt **alle** Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ **Phasen**

■ Innere Arbeit:

$$\mathcal{O} \left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil \right)$$

■ I/O Operationen:

$$\mathcal{O} \left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil \right)$$

■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

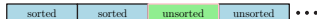
- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

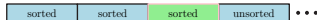
- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- alle Daten einmal lesen + schreiben

■ Multiway Merge

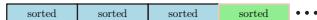
- jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- **alle** Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt **alle** Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- **alle** Daten einmal lesen + schreiben

■ Multiway Merge

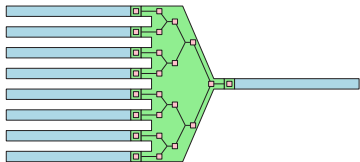
- jede Mischphase liest und schreibt **alle** Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- **alle** Daten einmal lesen + schreiben

■ Multiway Merge

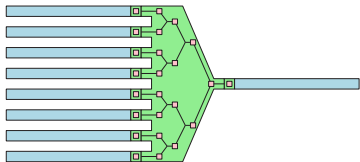
- jede Mischphase liest und schreibt **alle** Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O} \left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil \right)$$

■ I/O Operationen:

$$\mathcal{O} \left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil \right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

■ Run Formation

- Run entspricht einem Teilbereich zu sortierender Daten
- $\lceil \frac{n}{M} \rceil$ Stück, Größe M
- jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
- **alle** Daten einmal lesen + schreiben

■ Multiway Merge

- jede Mischphase liest und schreibt **alle** Daten $\rightarrow \frac{2n}{B}$ I/Os
- Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
- Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
 $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen

■ Innere Arbeit:

$$\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

■ I/O Operationen:

$$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

