

Übung 12 – 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

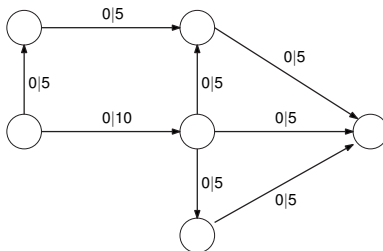
Institut für Theoretische Informatik - Algorithmik II

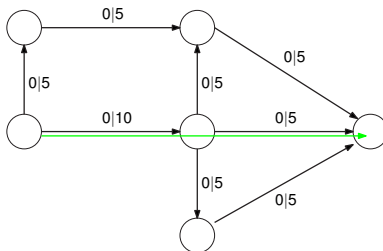
```
    result = current_weight;
    return true;
}

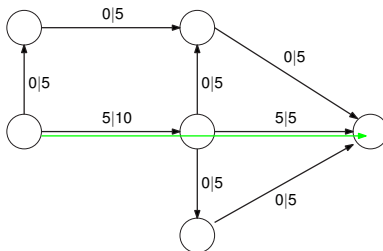
for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        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 );
            }
        }
    }
}
```

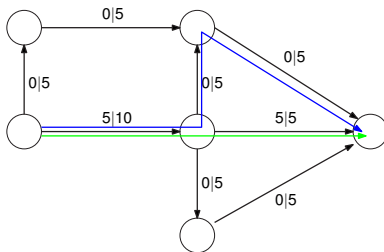
- Ford Fulkerson
 - Erhöhende Pfade
 - Residualgraph
- Max-Flow und Min-Cut
- Dinitz Algorithmus
 - Distanz Label
 - Layergraph
 - Blocking Flow

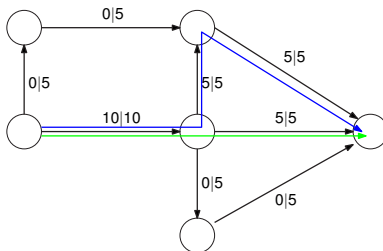
Ford Fulkerson

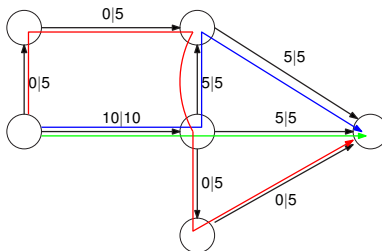


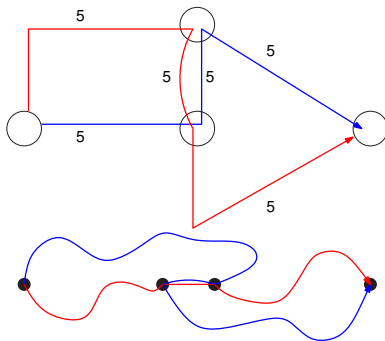


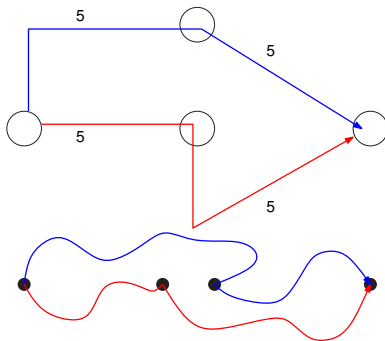


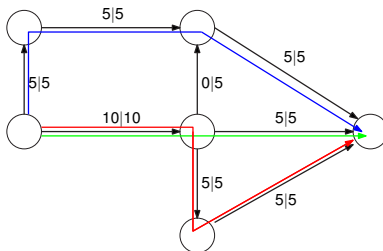




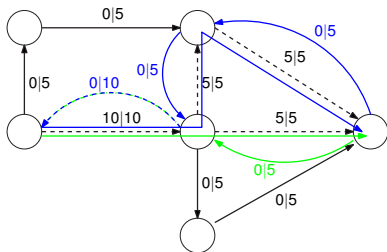




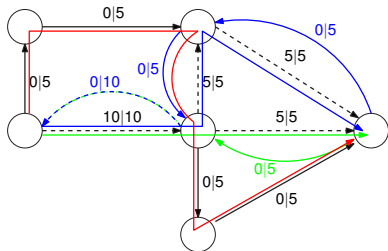
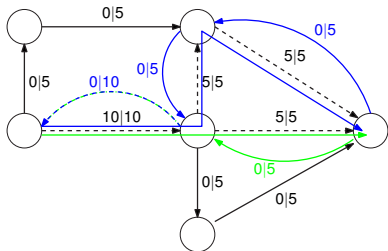




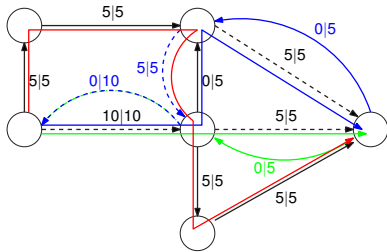
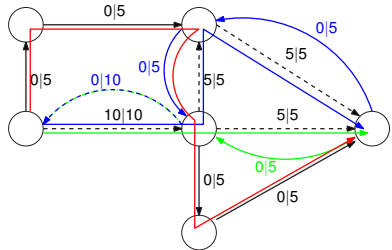
- Verwalten von Restkapazitäten
- Modellierung und Erkennung von “Gegenflüssen”
- $c^f(e) = c(e) - f(e)$: Restkapazität
Hier: Fluss $f(e)$ und Gesamtkapazität $c(e)$
- $c^f(e^{\text{rev}}) = f(e)$: Fluss über Kante e
Hier: Restkapazität und Gesamtkapazität von e
- Keine 0-Gewicht Kanten
- Flüsse über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



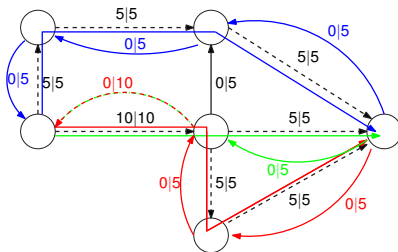
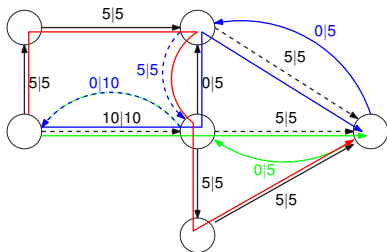
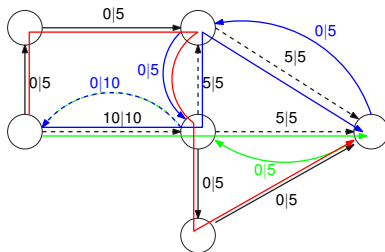
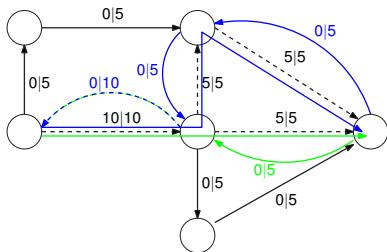
Flüsse und Ford Fulkerson



KIT
Karlsruher Institut für Technologie

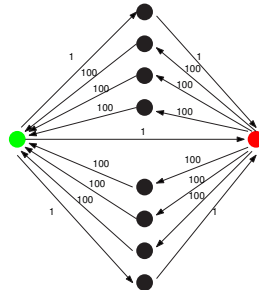


Flüsse und Ford Fulkerson



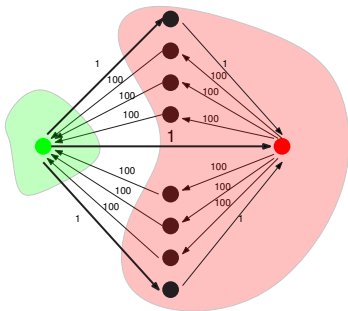
Max Flow - Min Cut

Max Flow - Min Cut

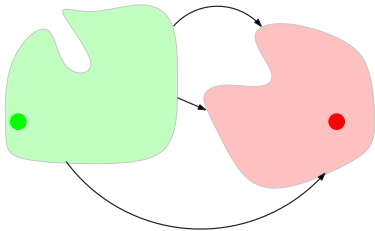


Max Flow - Min Cut

- S - T -Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S - T -Schnitte getrennt
- Flow muss von s nach t , auch durch alle möglichen S - T -Schnitte
→ **Max Flow = Min S - T -Cut**
- Cut und Flow mit gleichem Wert → beide optimal



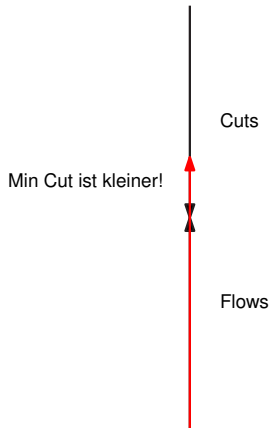
- S - T -Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S - T -Schnitte getrennt
- Flow muss von s nach t , auch durch alle möglichen S - T -Schnitte
→ **Max Flow = Min S - T -Cut**
- Cut und Flow mit gleichem Wert → beide optimal



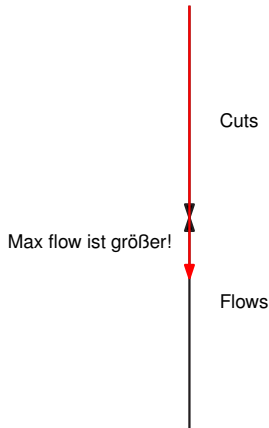
- S - T -Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S - T -Schnitte getrennt
- Flow muss von s nach t , auch durch alle möglichen S - T -Schnitte
→ **Max Flow = Min S - T -Cut**
- Cut und Flow mit gleichem Wert → beide optimal



- S - T -Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S - T -Schnitte getrennt
- Flow muss von s nach t , auch durch alle möglichen S - T -Schnitte
→ **Max Flow = Min S - T -Cut**
- Cut und Flow mit gleichem Wert → beide optimal

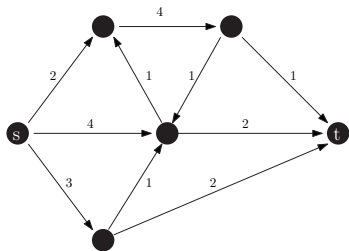


- S - T -Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- s und t werden durch alle möglichen S - T -Schnitte getrennt
- Flow muss von s nach t , auch durch alle möglichen S - T -Schnitte
→ **Max Flow = Min S - T -Cut**
- Cut und Flow mit gleichem Wert → beide optimal

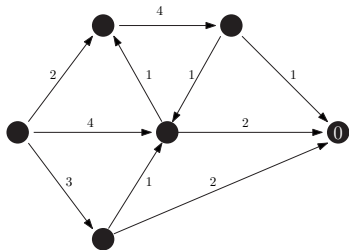


Dinitz Algorithmus

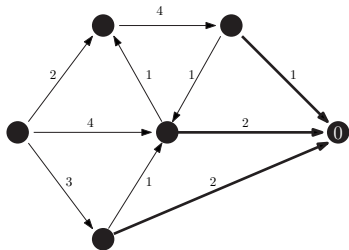
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



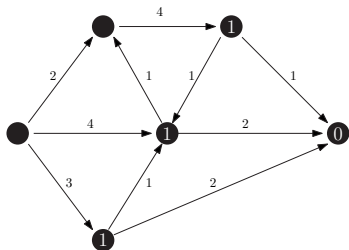
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



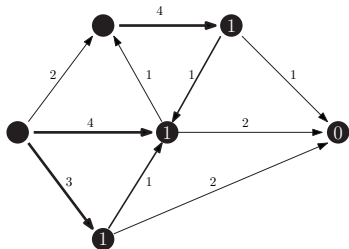
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



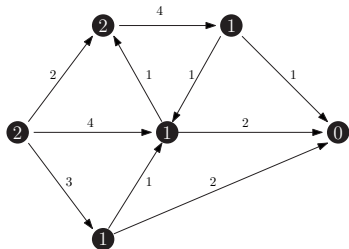
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
auch: kürzeste Wege Netzwerk, Schichtgraph



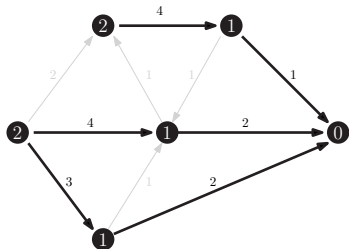
- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
 - Start bei Knoten t
 - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
 - Knoten in einem Layer: gleiches Label
- *Layered graph*
 auch: kürzeste Wege Netzwerk, Schichtgraph



- 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 $E_S = E' \cup E'^{rev}$



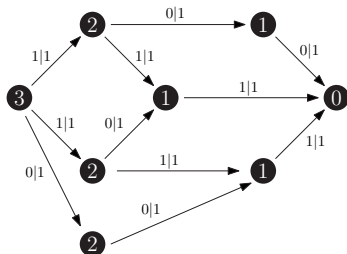
■ Kein weiterer Fluss möglich

“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



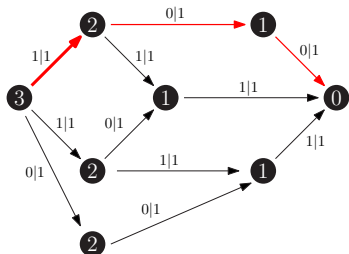
■ Kein weiterer Fluss möglich

“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



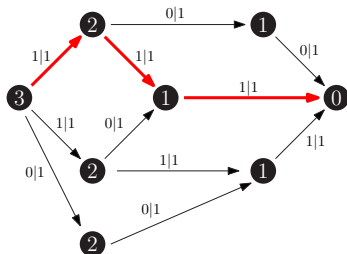
■ Kein weiterer Fluss möglich

“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



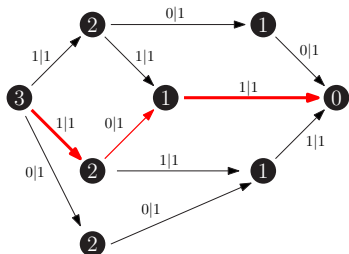
■ Kein weiterer Fluss möglich

“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



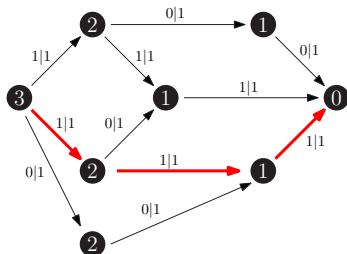
■ Kein weiterer Fluss möglich

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

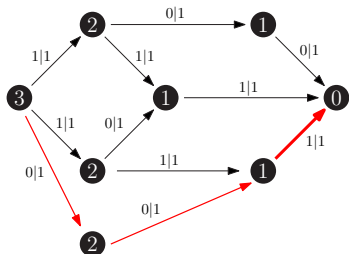
■ *Blocking flow als atomare Operation*

- Berechnung auf Schichtgraph
- Kein Residualgraph
- Kein Rückfluss möglich

■ i.A. nicht maximal auf Schichtgraph und Residualgraph

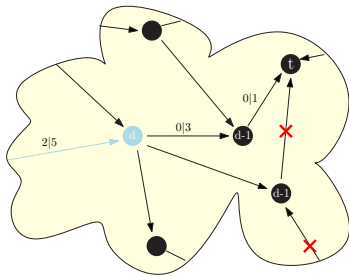


- Kein weiterer Fluss möglich
 “Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist”
- *Blocking flow als atomare Operation*
 - Berechnung auf Schichtgraph
 - Kein Residualgraph
 - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Residualgraph

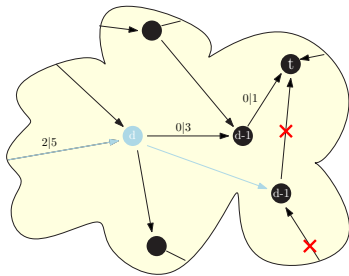


- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
 - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten

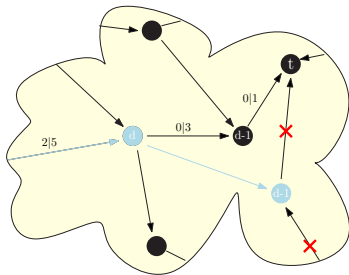
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



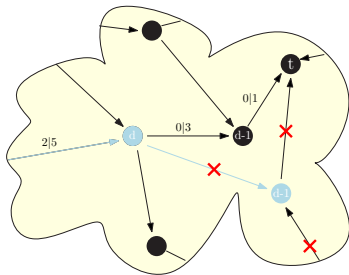
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



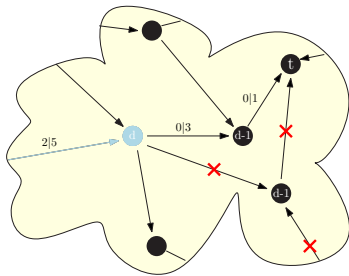
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



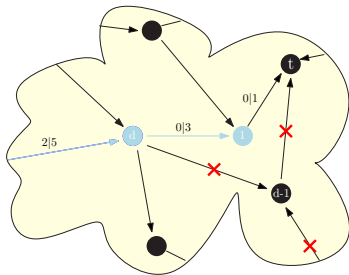
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



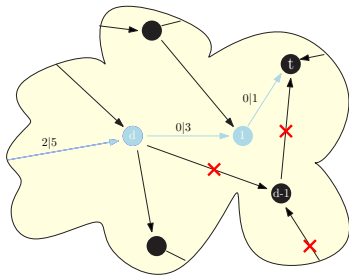
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



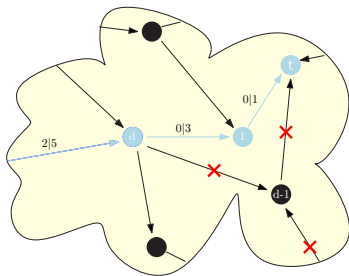
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



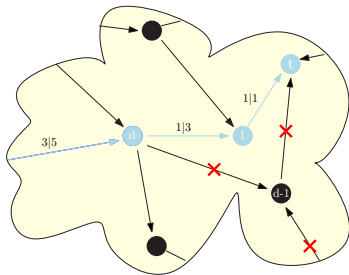
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



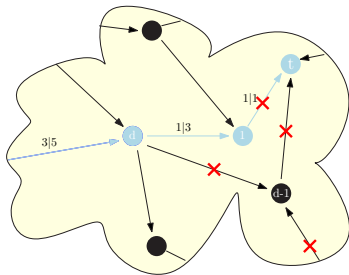
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
 - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
 - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
 - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



- *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 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



- $\#breakthrough \leq m$
 - 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 $\leq n$

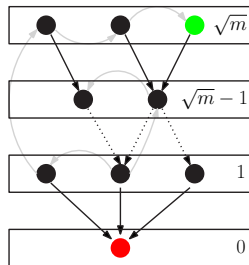
(Lemma 1: Jede Phase erhöht Label von s um mindestens 1)

→ **Gesamtlaufzeit** $O(n^2m)$

Amortisierte Kosten

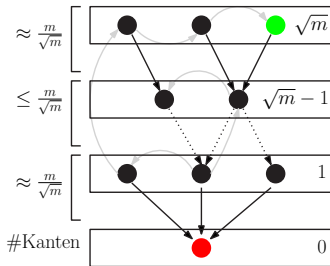
- Retreat/breakthrough
 - Löscht alle beteiligten Kanten
→ Jede Kante nur an einem Retreat/breakthrough beteiligt
- Extend
 - Pro Kante ein *extend*
 - Beahlt für Breakthrough und Retreat
 - #extends = $O(m)$
 - Kosten $O(1)$
- Phase in $O(m + n)$

- 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**



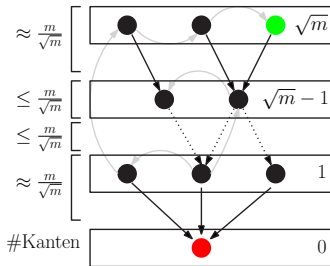
■ Nach $\sqrt{m}$ Phasen

- 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**



Ende!



Feierabend!