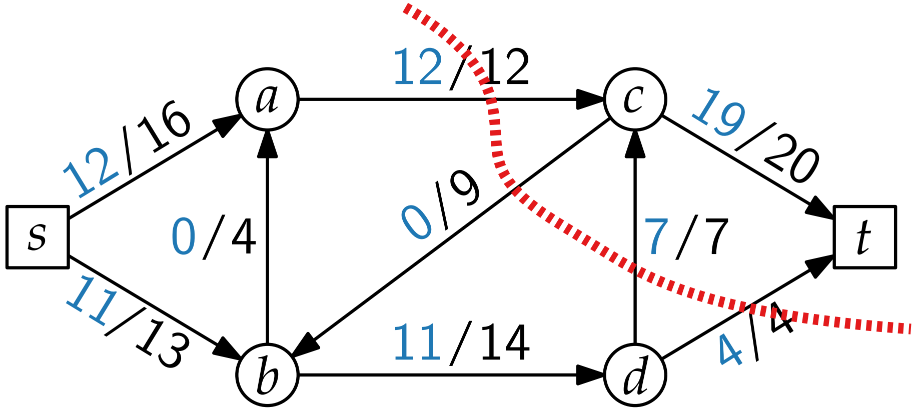
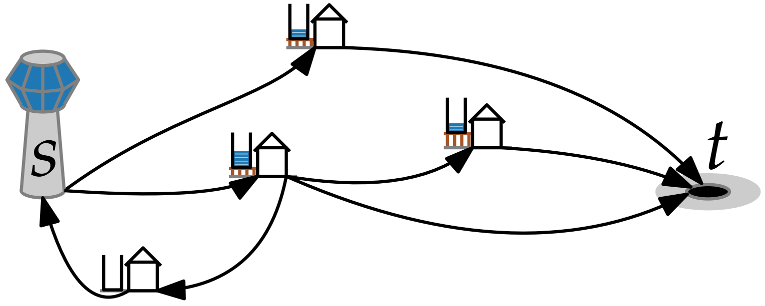
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Maximum Flow Problem

Push–Relabel Algorithm

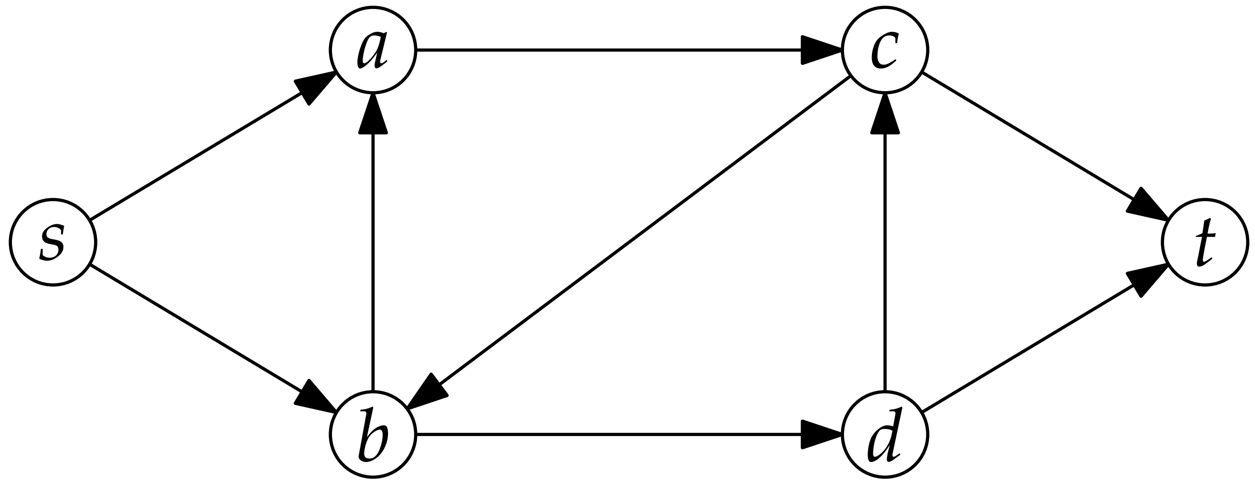




Alexander Wolff

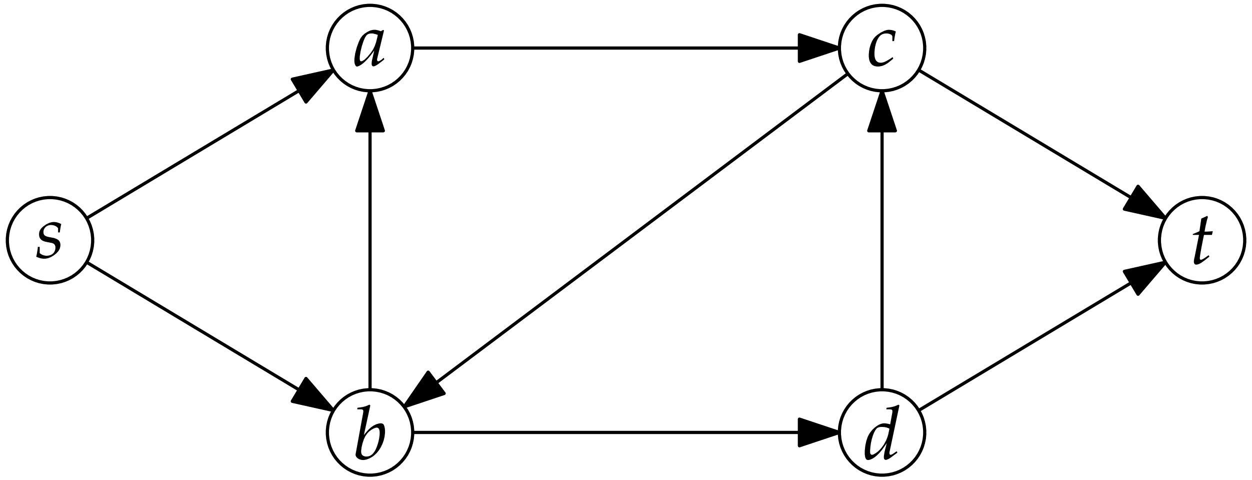
Summer term 2026

Flow Networks



Flow Networks

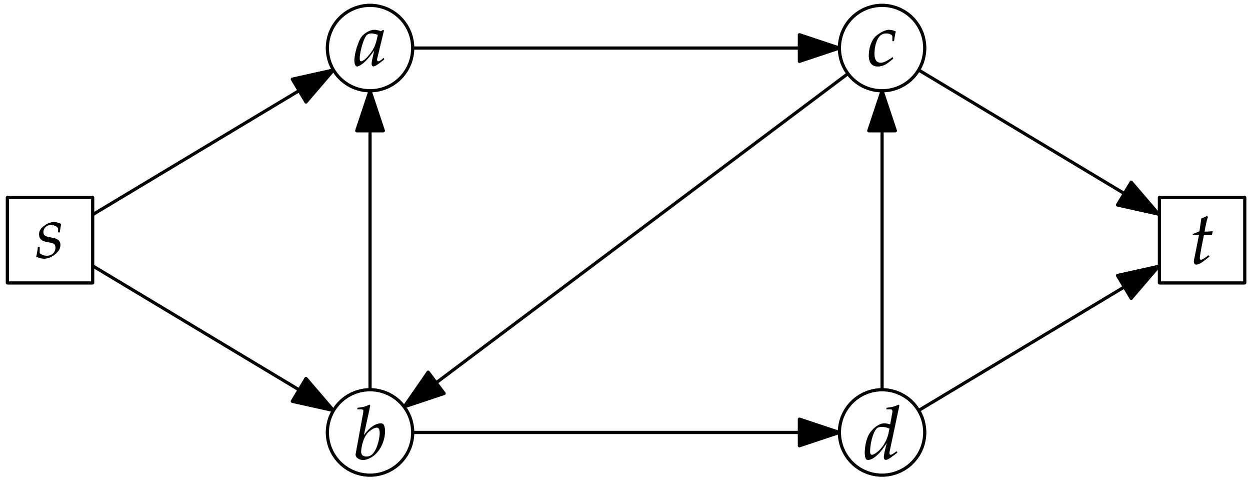
A **flow network** G is a digraph (short for "**di**rected **graph**") with



Flow Networks

A **flow network** G is a digraph (short for "**di**rected **graph**") with

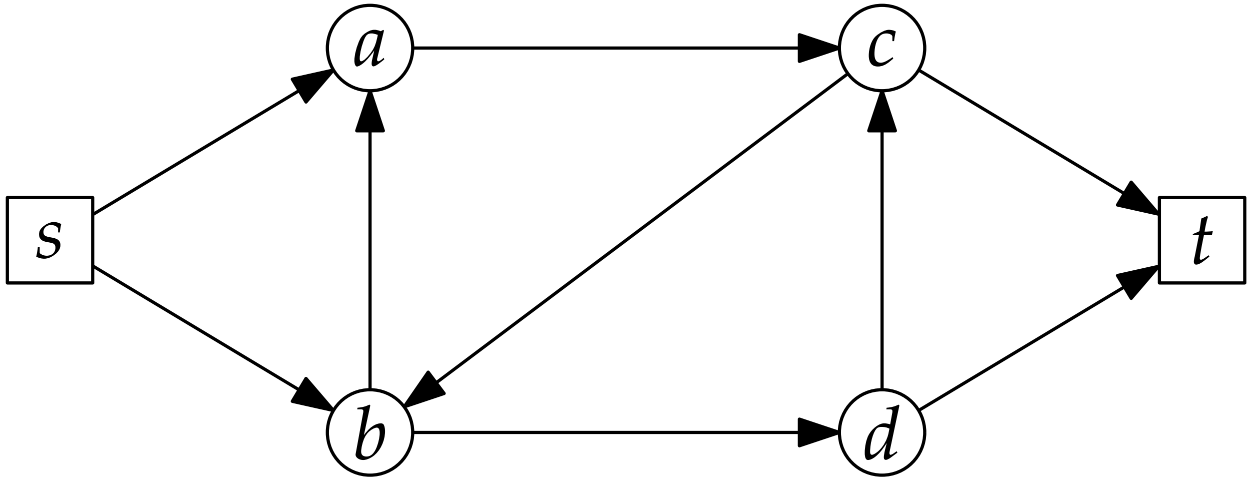
- unique **source** s and **sink** t,



Flow Networks

A **flow network** G is a digraph (short for "**di**rected **graph**") with

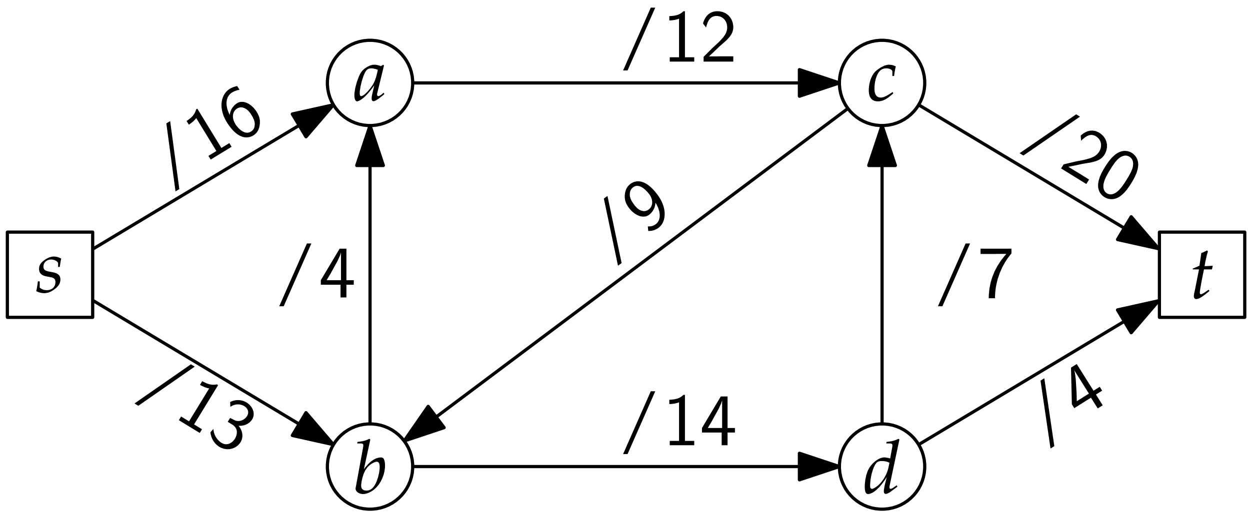
- unique **source** s and **sink** t,
- no antiparallel edges, and



Flow Networks

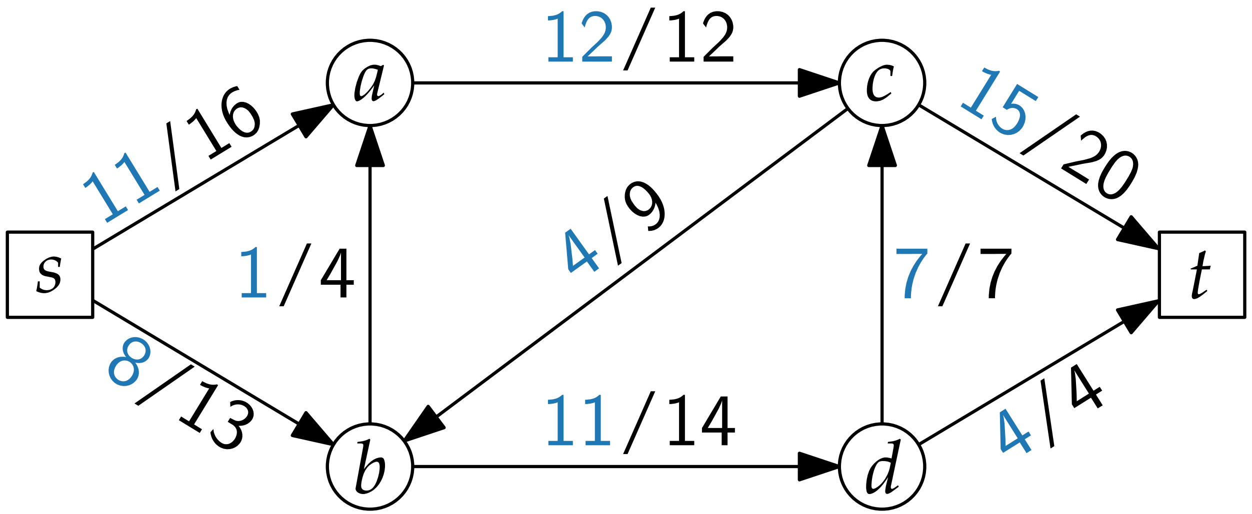
A **flow network** G is a digraph (short for "**di**rected **graph**") with

- unique **source** s and **sink** t,
- no antiparallel edges, and
- a **capacity** $c(u, v) \geq 0$ for every $(u, v) \in E(G)$.



Flow

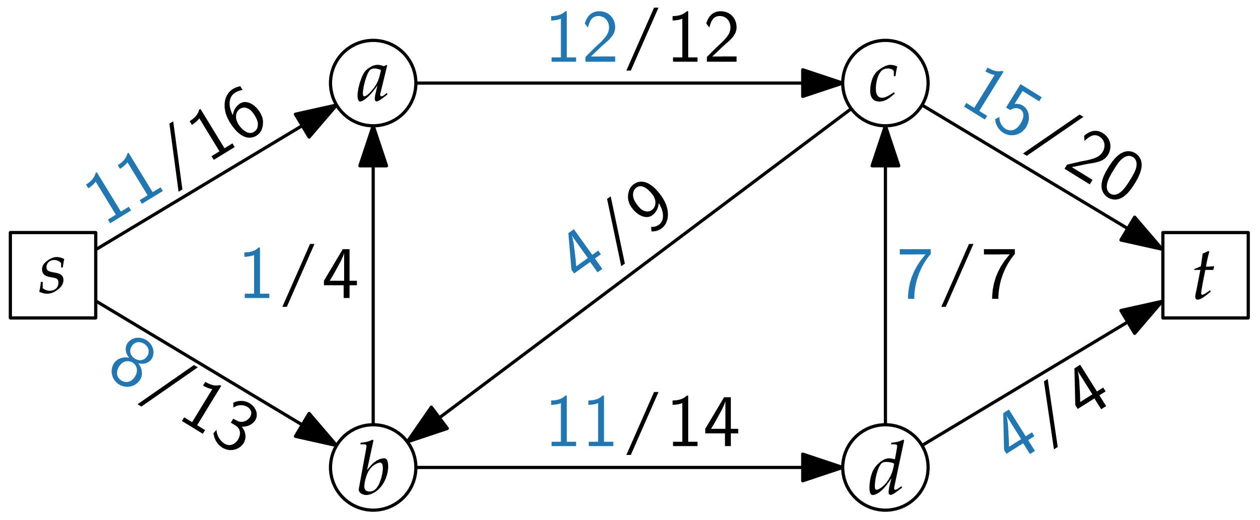
An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies



Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

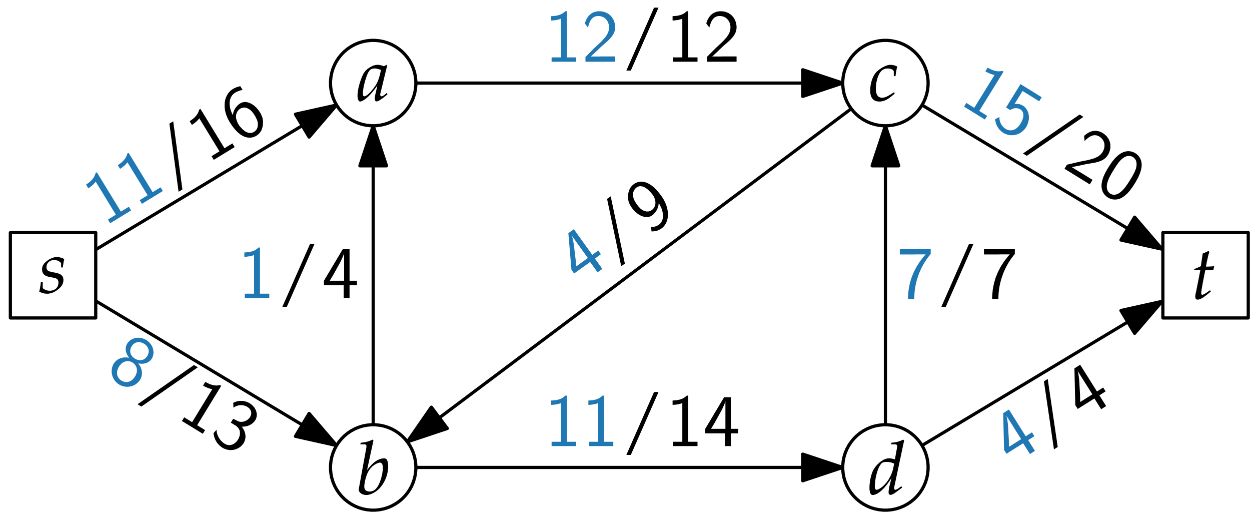
- **flow conservation**:
  $\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v)$ for every $u \in V(G) \setminus \{s,t\}$, and



Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

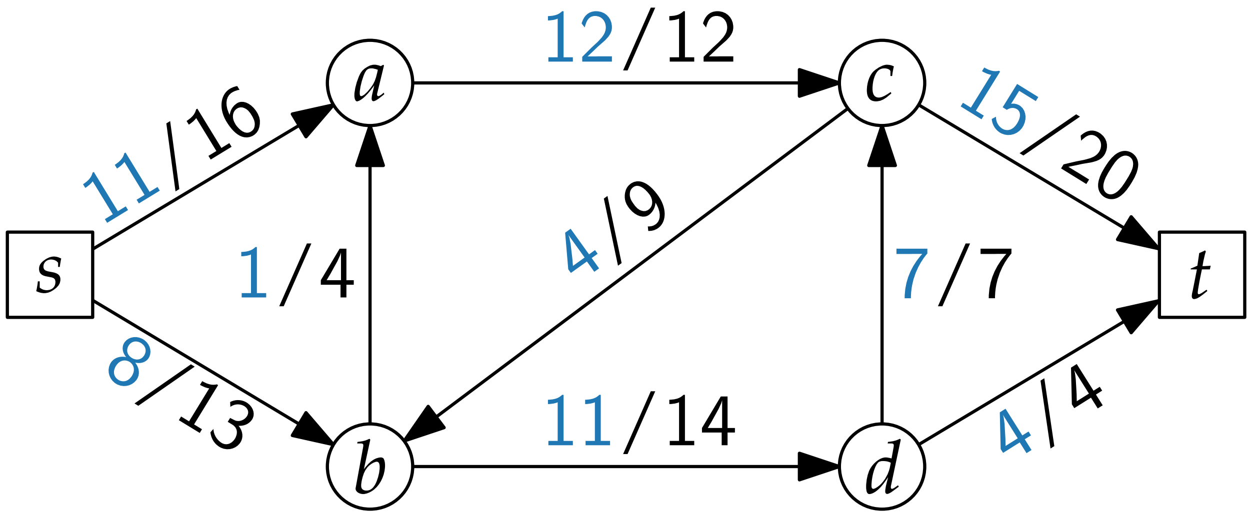
- **flow conservation**:
  $$\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v) \text{ for every } u \in V(G) \setminus \{s,t\}, \text{ and}$$
- **capacity constraints**: $0 \le f(u,v) \le c(u,v)$ for every $(u,v) \in E(G)$.



Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

- **flow conservation**:
  $\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v)$ for every $u \in V(G) \setminus \{s,t\}$, and
- **capacity constraints**: $0 \le f(u,v) \le c(u,v)$ for every $(u,v) \in E(G)$.



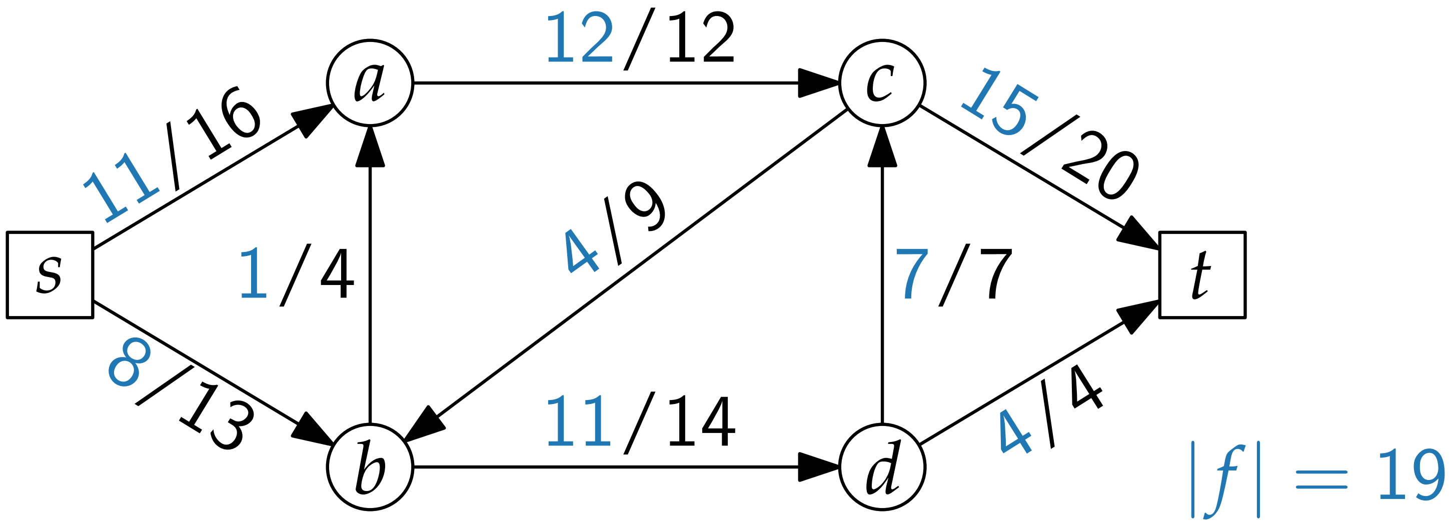
The **value** $|f|$ of an s–t flow f is defined as

$$|f| = \sum_{v \in N^-(s)} f(s,v) - \sum_{v \in N^+(s)} f(v,s).$$

Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

- **flow conservation**:
  $\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v)$ for every $u \in V(G) \setminus \{s,t\}$, and
- **capacity constraints**: $0 \le f(u,v) \le c(u,v)$ for every $(u,v) \in E(G)$.



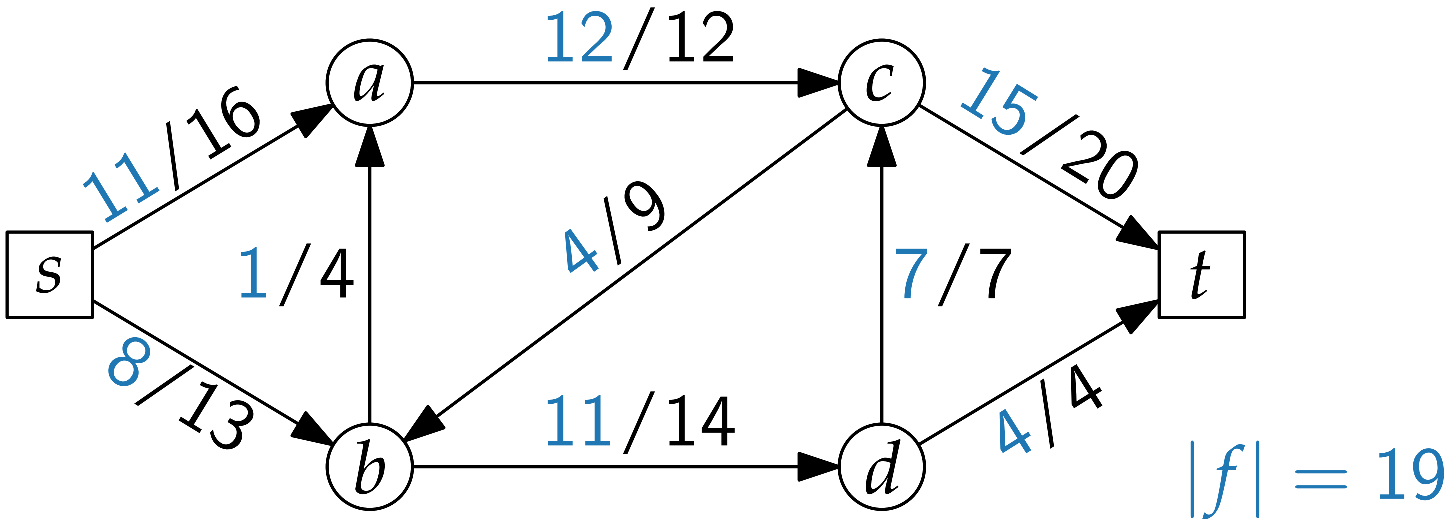
The **value** $|f|$ of an s–t flow f is defined as

$$|f| = \sum_{v \in N^-(s)} f(s,v) - \sum_{v \in N^+(s)} f(v,s).$$

Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

- **flow conservation**:
  $\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v)$ for every $u \in V(G) \setminus \{s,t\}$, and
- **capacity constraints**: $0 \le f(u,v) \le c(u,v)$ for every $(u,v) \in E(G)$.



Maximum flow problem. Given a flow network G with source s and sink t, find an s–t flow of maximum value.

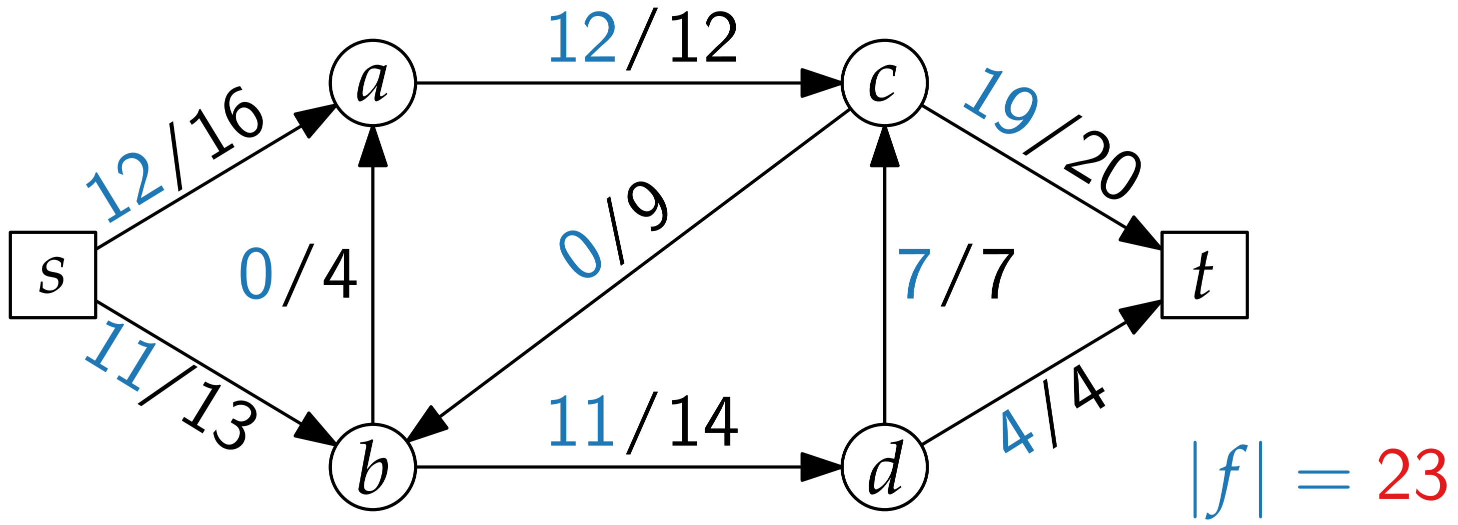
The **value** $|f|$ of an s–t flow f is defined as

$$|f| = \sum_{v \in N^-(s)} f(s,v) - \sum_{v \in N^+(s)} f(v,s).$$

Flow

An s–t **flow** in G is a real-valued function $f\colon E(G) \to \mathbb{R}$ that satisfies

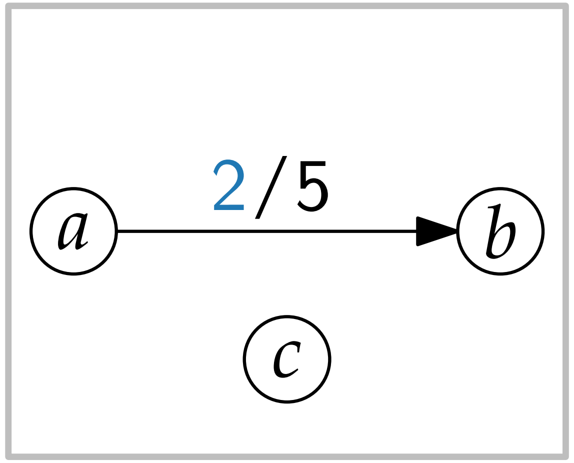
- **flow conservation**:
  $$\sum_{v \in N^+(u)} f(v,u) = \sum_{v \in N^-(u)} f(u,v) \text{ for every } u \in V(G) \setminus \{s,t\}, \text{ and}$$
- **capacity constraints**: $0 \le f(u,v) \le c(u,v)$ for every $(u,v) \in E(G)$.



Maximum flow problem.
Given a flow network G with source s and sink t, find an s–t flow of maximum value.

The **value** $|f|$ of an s–t flow f is defined as
$$|f| = \sum_{v \in N^-(s)} f(s,v) - \sum_{v \in N^+(s)} f(v,s).$$

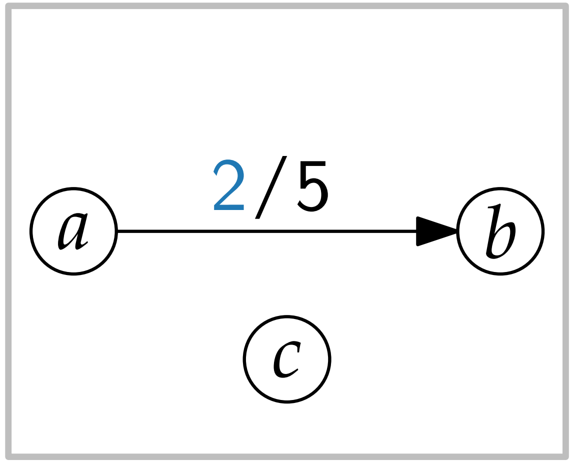
By How Much May Flow Change?



By How Much May Flow Change?

Given G and f, the **residual capacity** c_f for a pair $u, v \in V(G)$ is

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E(G) \\ f(v,u) & \text{if } (v,u) \in E(G) \\ 0 & \text{otherwise.} \end{cases}$$



$c_f(a,b) = 3$

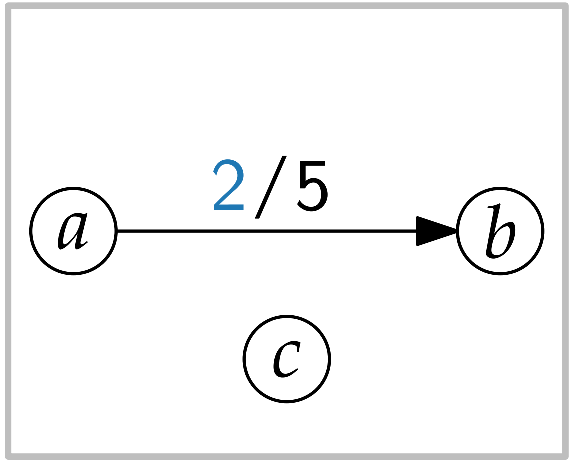
$c_f(b,a) = 2$

$c_f(a,c) = 0$

By How Much May Flow Change?

Given G and f, the **residual capacity** c_f for a pair $u, v \in V(G)$ is

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E(G) \\ f(v,u) & \text{if } (v,u) \in E(G) \\ 0 & \text{otherwise.} \end{cases}$$

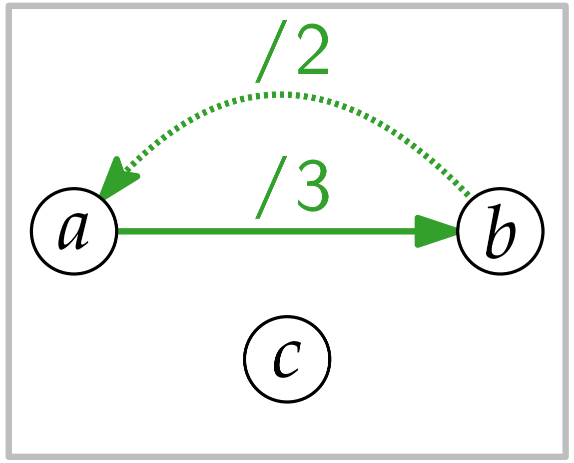


flow network G

$c_f(a,b) = 3$

$c_f(b,a) = 2$

$c_f(a,c) = 0$



residual network G_f

Residual Networks & Augmenting Paths

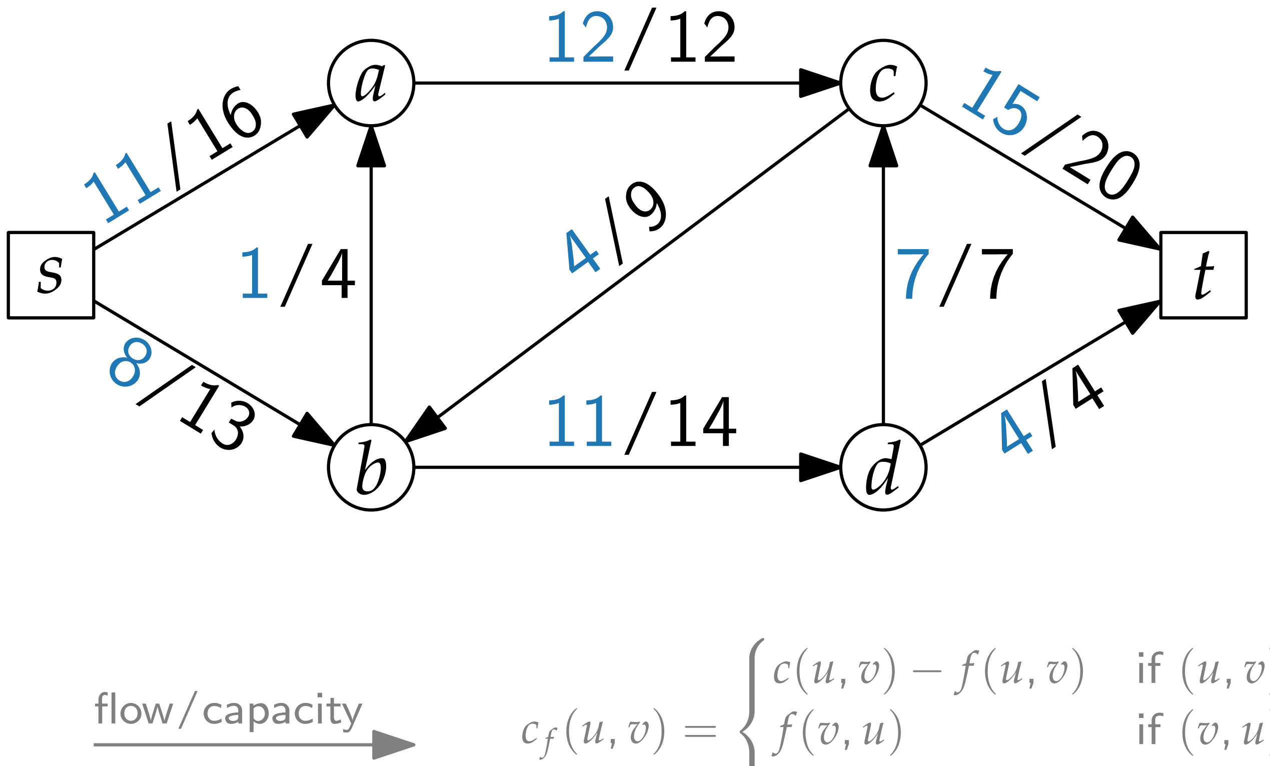
The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u, v) \in V(G)^2 \mid c_f(u, v) > 0\}$.

Residual Networks & Augmenting Paths

The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.

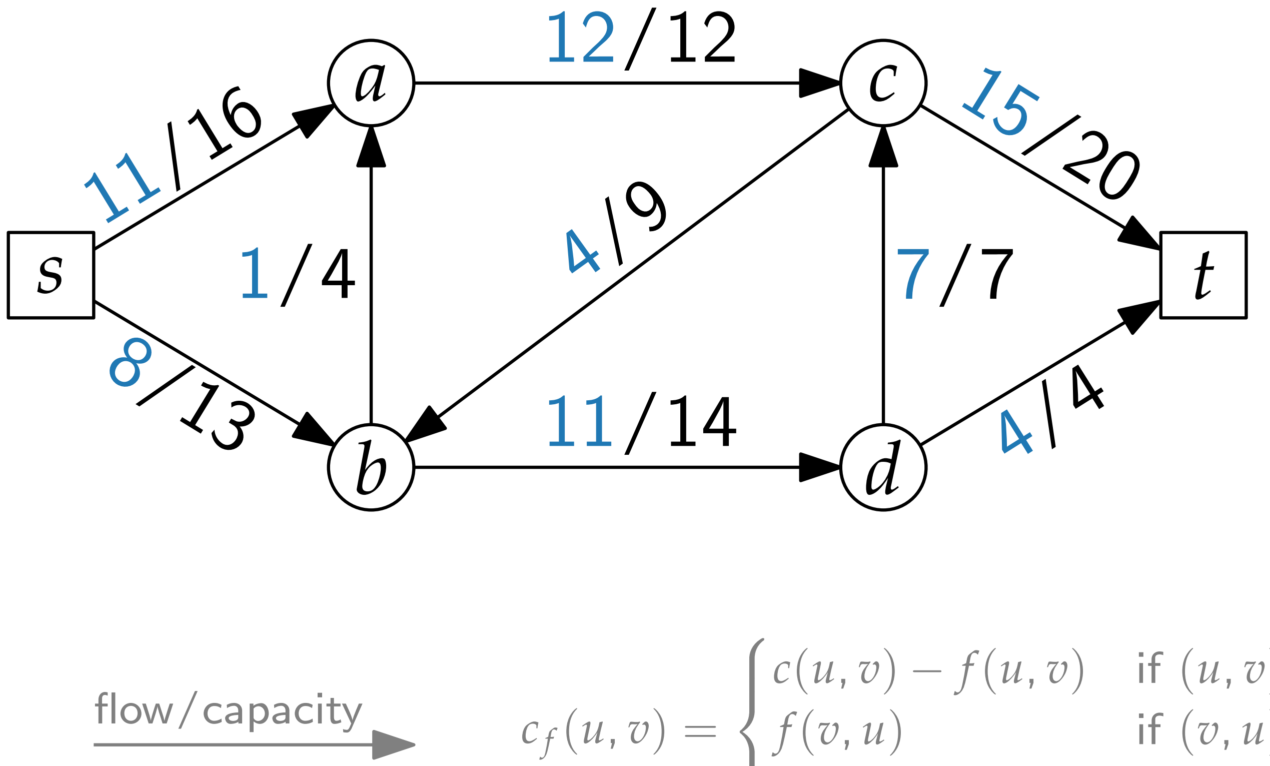


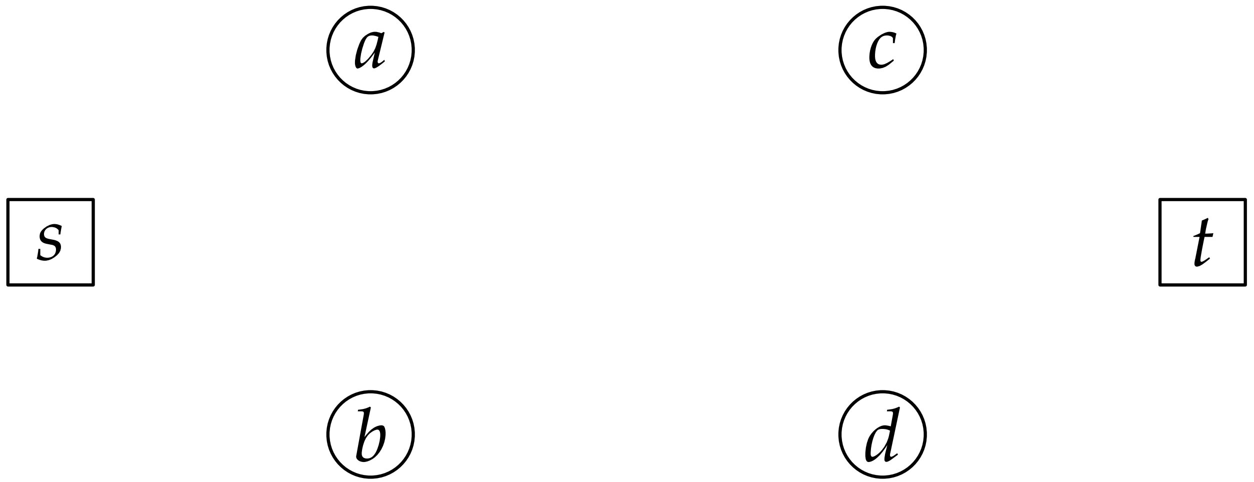
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Residual Networks & Augmenting Paths

The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.



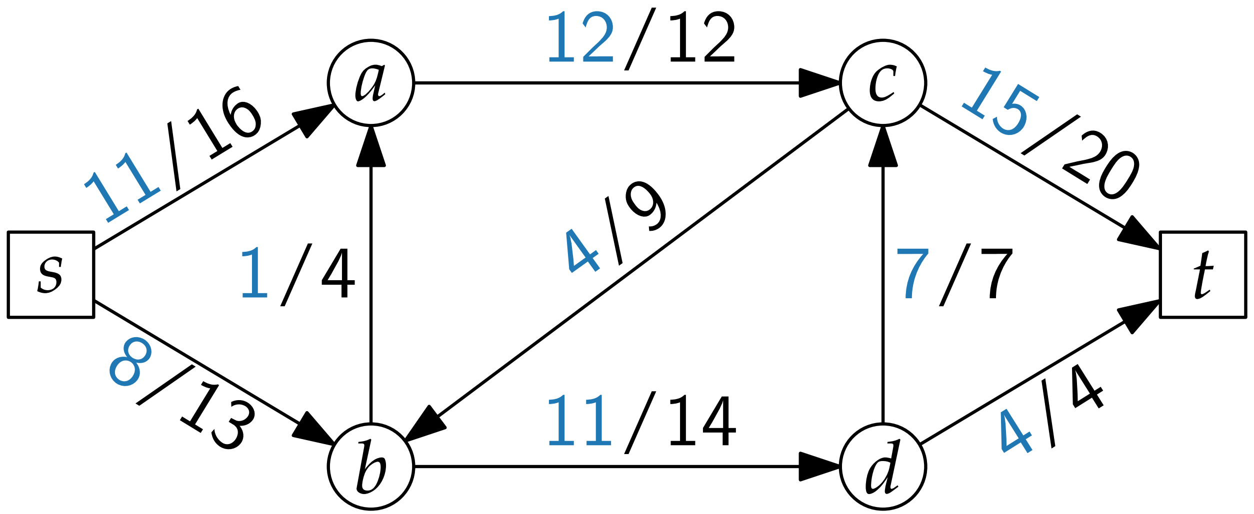


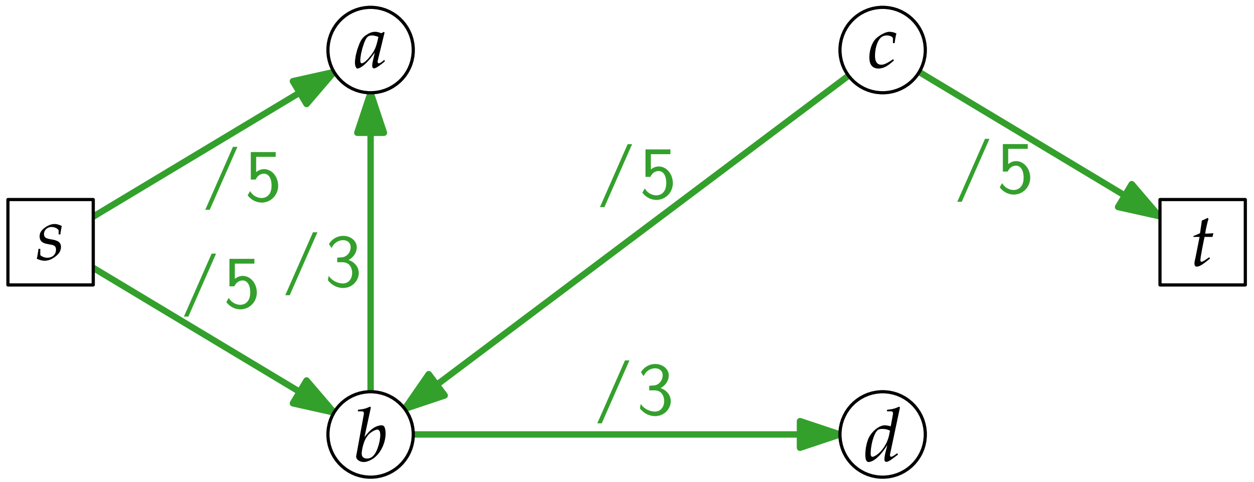
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Residual Networks & Augmenting Paths

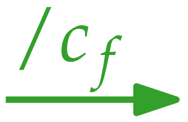
The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.





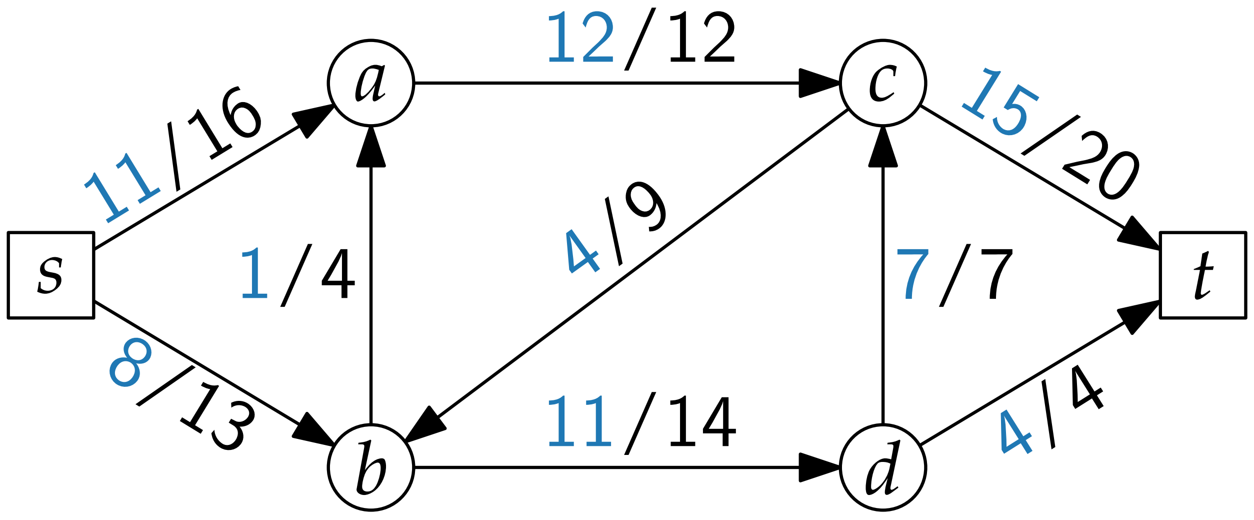
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$

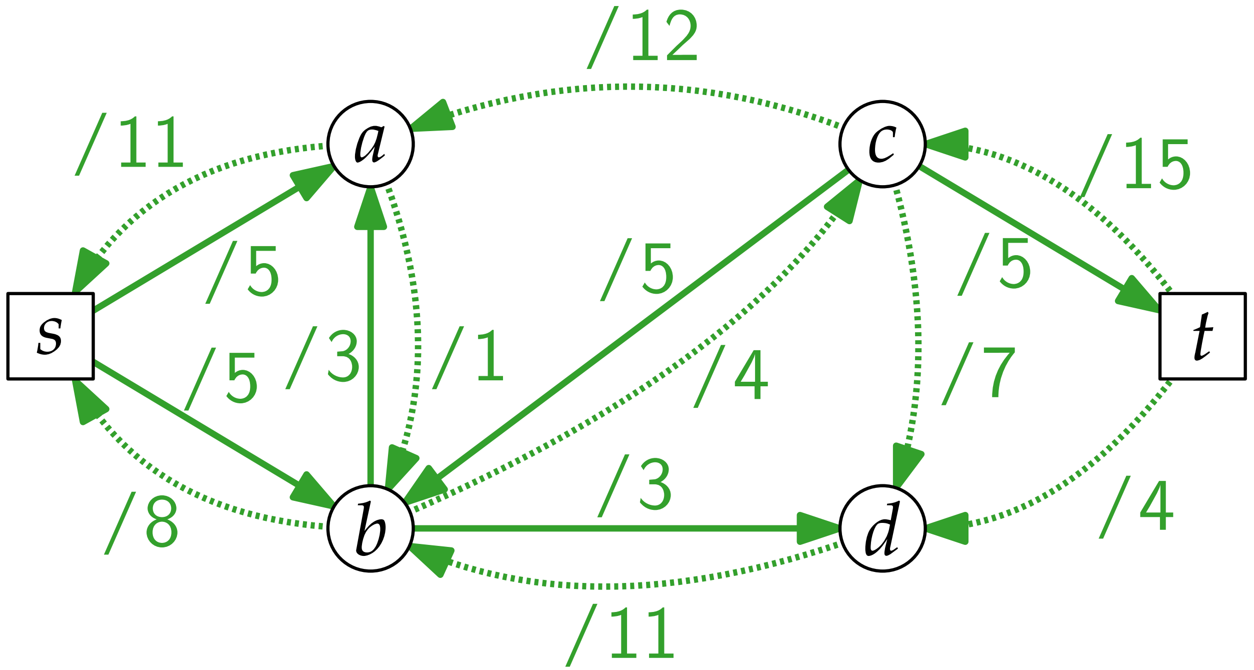


Residual Networks & Augmenting Paths

The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.





flow/capacity

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$

$/c_f$ non-satured edges

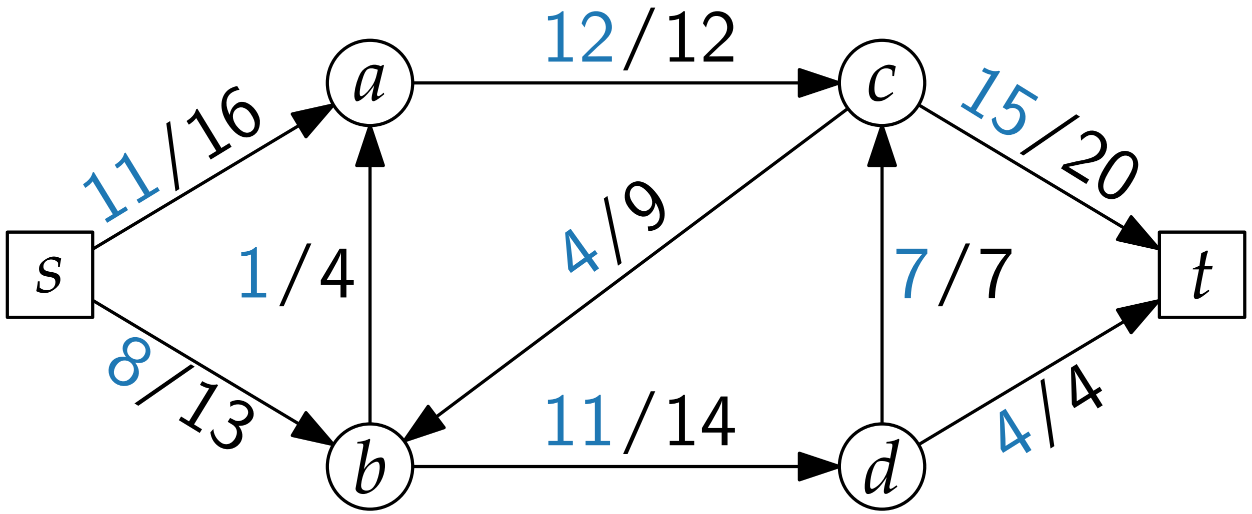
reverse edges

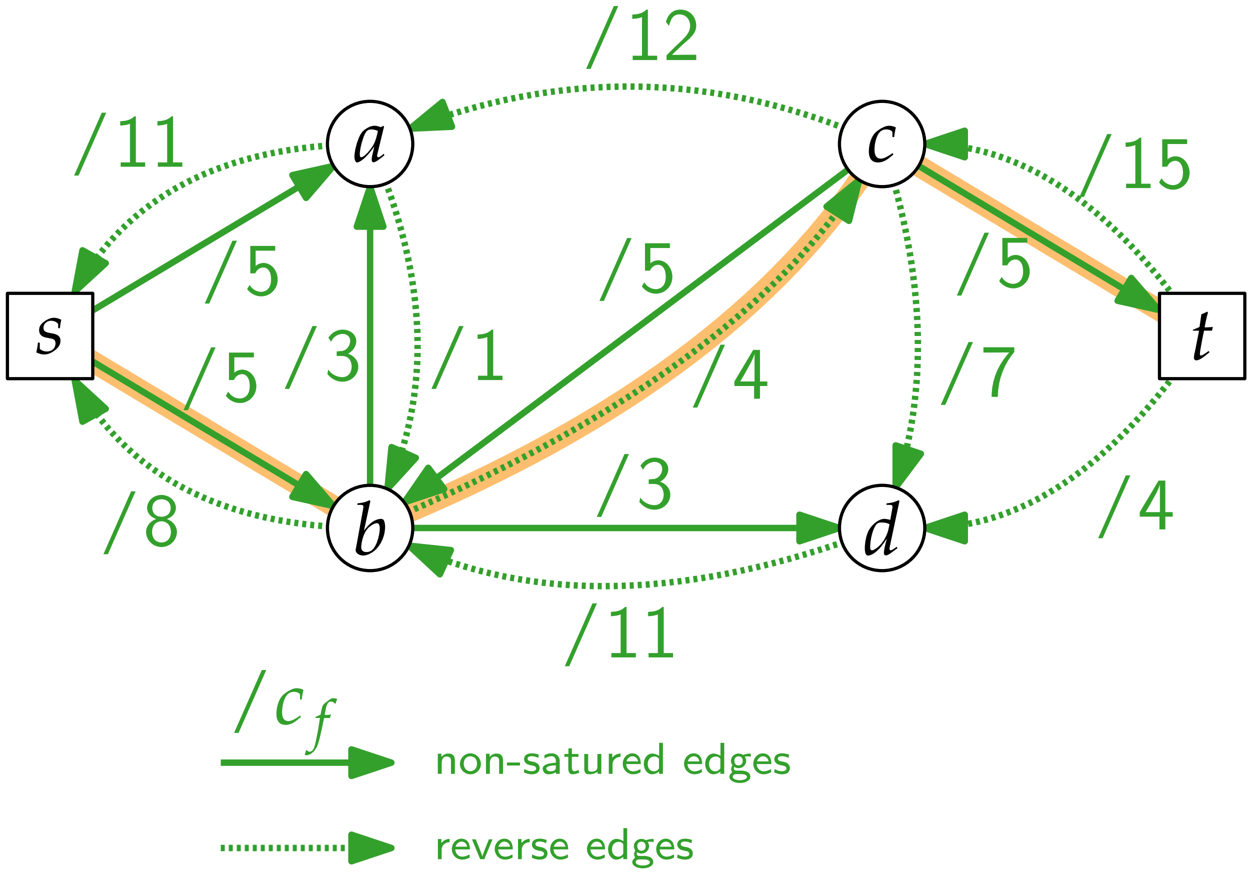
Residual Networks & Augmenting Paths

The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.

An **augmenting path** is an s–t path in G_f





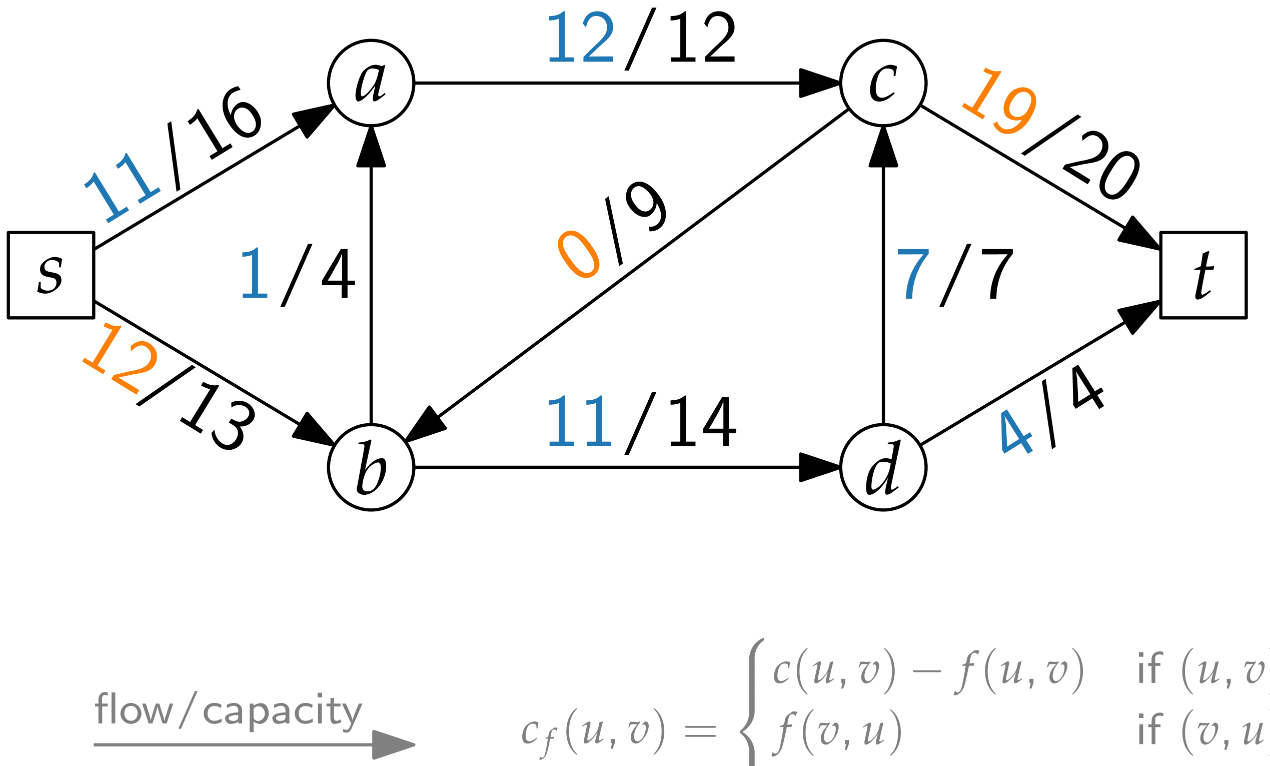
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Residual Networks & Augmenting Paths

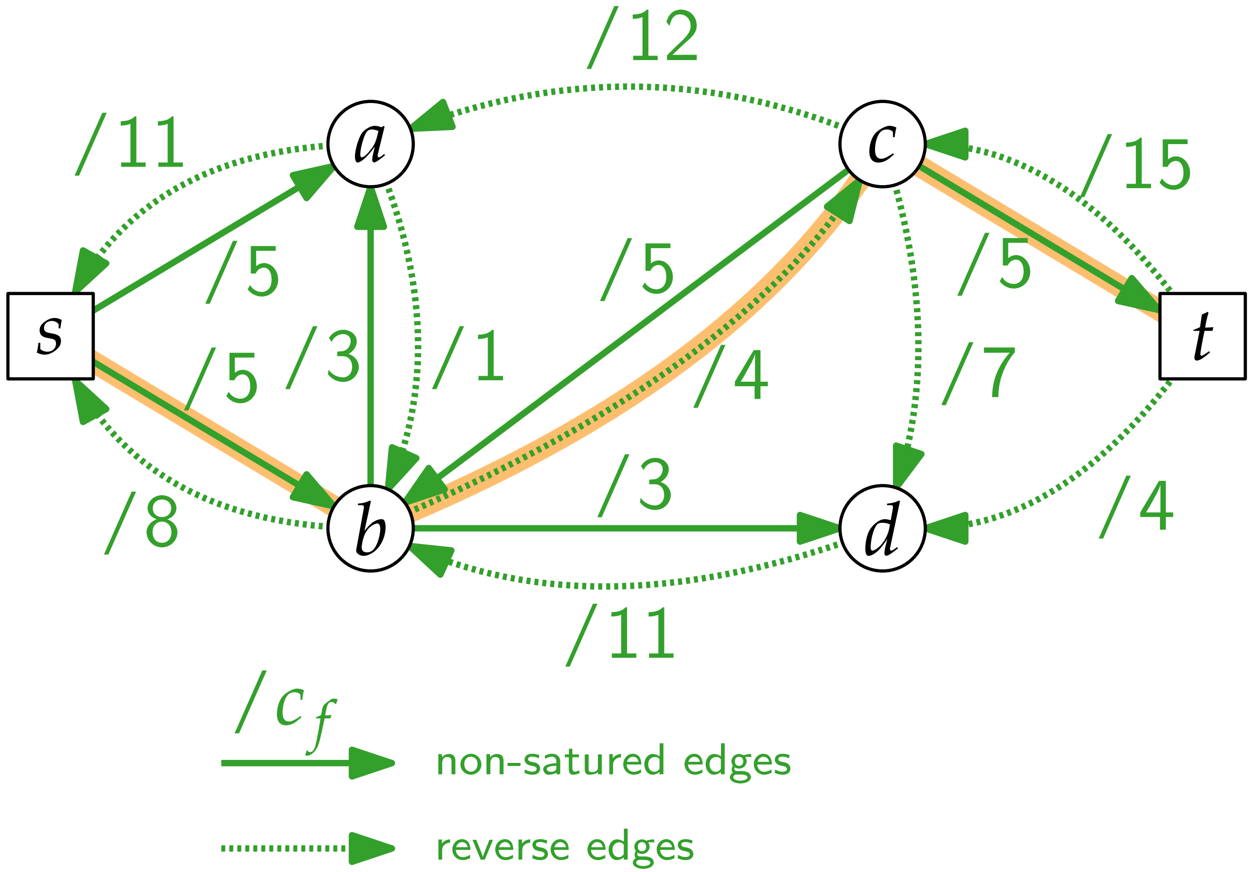
The **residual network** G_f for a flow network G with s–t flow f has

- $V(G_f) = V(G)$ and $E(G_f) = \{(u,v) \in V(G)^2 \mid c_f(u,v) > 0\}$.

An **augmenting path** is an s–t path in G_f $\Rightarrow$ use to increase f.



$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise.} \end{cases}$$



The Algorithms of Ford–Fulkerson and Edmonds–Karp

```
FordFulkerson(G, c, s, t):
  foreach (u, v) ∈ E(G) do                          } initializing zero flow
  └ f(u, v) = 0
  while G_f contains augmenting path p do
  │  Δ = min_{(u,v)∈p} c_f(u, v)                     } residual capacity of p
  │  foreach (u, v) ∈ p do                           ⎫
  │  │  if (u, v) ∈ E(G) then                        ⎪
  │  │  │  f(u, v) = f(u, v) + Δ                     ⎬ augmentation along p
  │  │  else                                         ⎪
  │  │  └ f(v, u) = f(v, u) − Δ                      ⎭
  │  └
  └
  return f                                           } return max flow
```

The Algorithms of Ford–Fulkerson and Edmonds–Karp

EdmondsKarp

~~FordFulkerson~~(G, c, s, t):

foreach $(u, v) \in E(G)$ **do**
  $f(u, v) = 0$
} initializing zero flow

while G_f contains shortest augmenting path p **do**
  $\Delta = \min_{(u,v)\in p} c_f(u, v)$ } residual capacity of p
  foreach $(u, v) \in p$ **do**
    if $(u, v) \in E(G)$ **then**
      $f(u, v) = f(u, v) + \Delta$
    else
      $f(v, u) = f(v, u) - \Delta$
  } augmentation along p

return f } return max flow

The Algorithms of Ford–Fulkerson and Edmonds–Karp

EdmondsKarp

~~FordFulkerson~~(G, c, s, t):

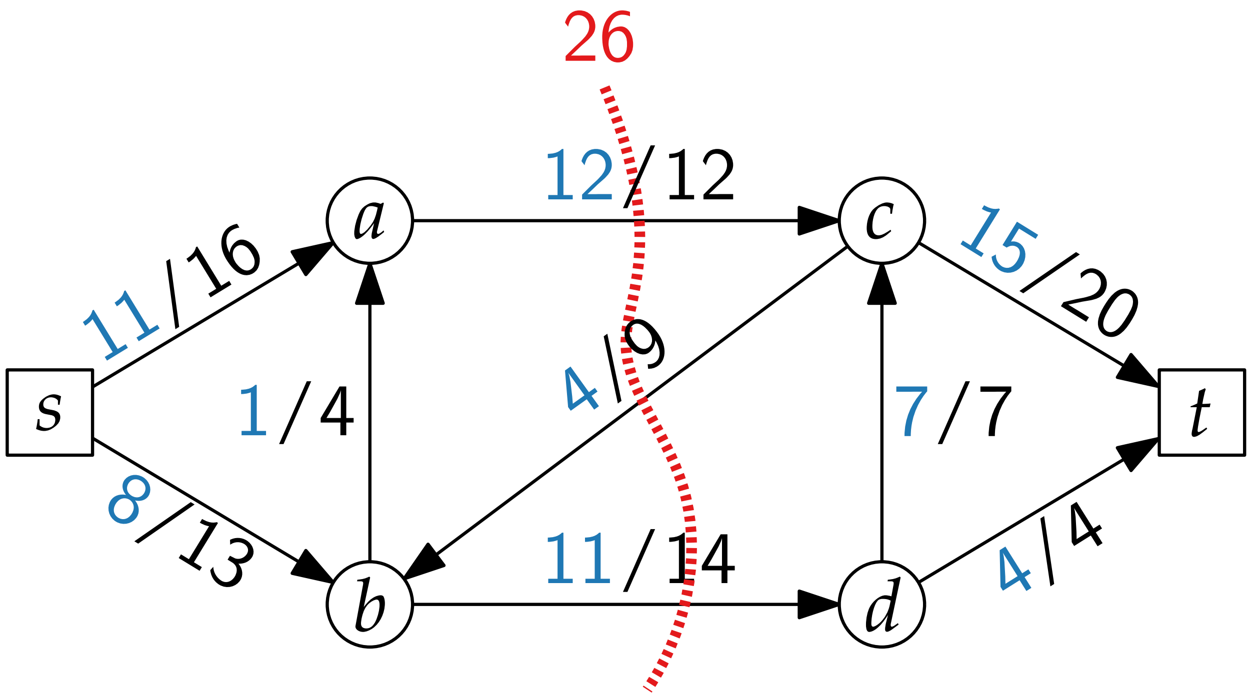
foreach $(u, v) \in E(G)$ **do**
  $f(u, v) = 0$ } initializing zero flow

while G_f contains shortest augmenting path p **do**
  $\Delta = \min_{(u,v) \in p} c_f(u, v)$ } residual capacity of p
  foreach $(u, v) \in p$ **do**
    if $(u, v) \in E(G)$ **then**
      $f(u, v) = f(u, v) + \Delta$
    else
      $f(v, u) = f(v, u) - \Delta$
  } augmentation along p

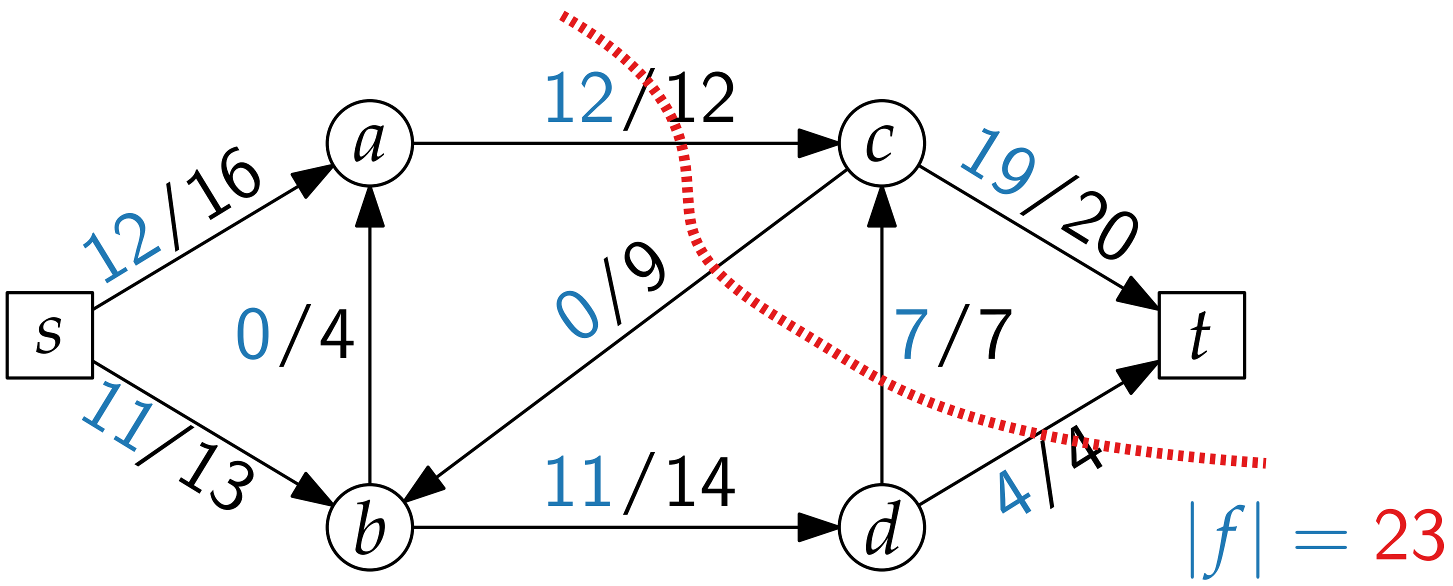
return f } return max flow

- Ford–Fulkerson runs in $\mathcal{O}(|E||V| \max_{e \in E} c(e))$; Edmonds–Karp in $\mathcal{O}(|E|^2|V|)$ time.

The Max-Flow Min-Cut Theorem



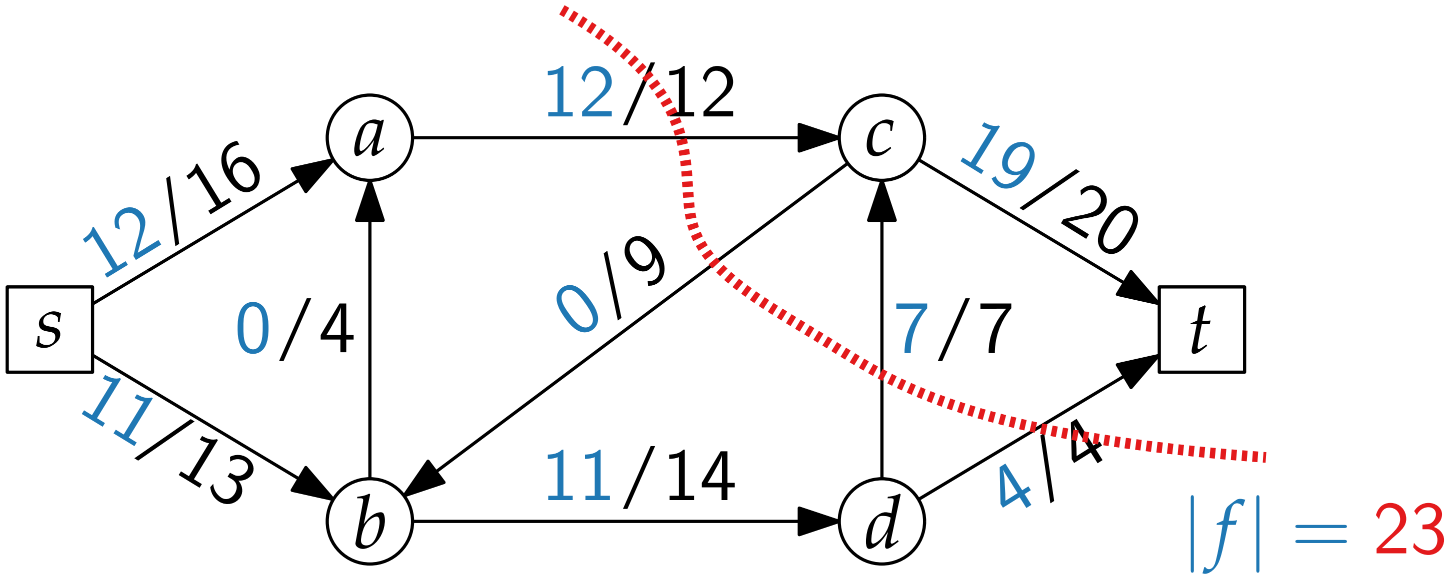
The Max-Flow Min-Cut Theorem



The Max-Flow Min-Cut Theorem

Theorem.
For an s–t flow f in a flow network G, the following conditions are equivalent:

- f is a maximum s–t flow in G.
- G_f contains no augmenting paths.
- $|f| = c(S,T)$, which is the capacity of at least one s–t cut (S,T) of G.



The Push–Relabel Idea

A New Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A New Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the t[illegible]al amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

[1988]

A (New) Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

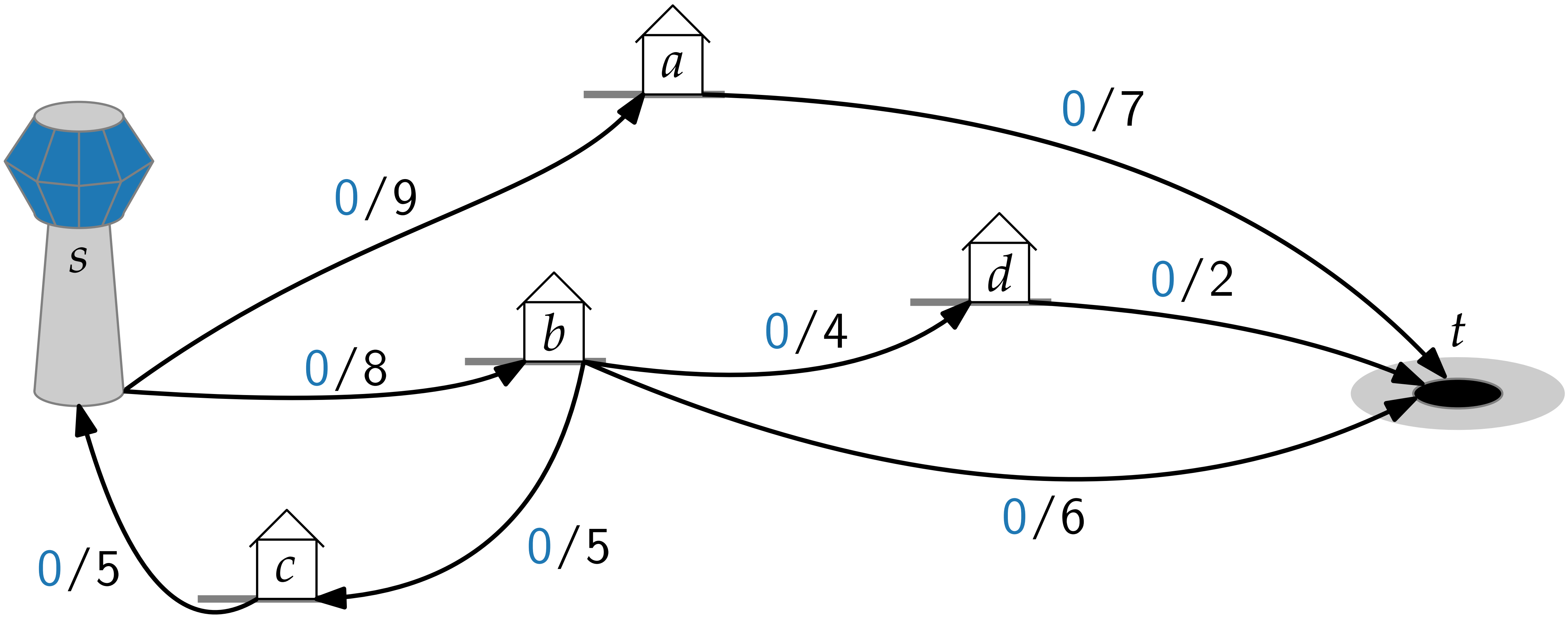
Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount

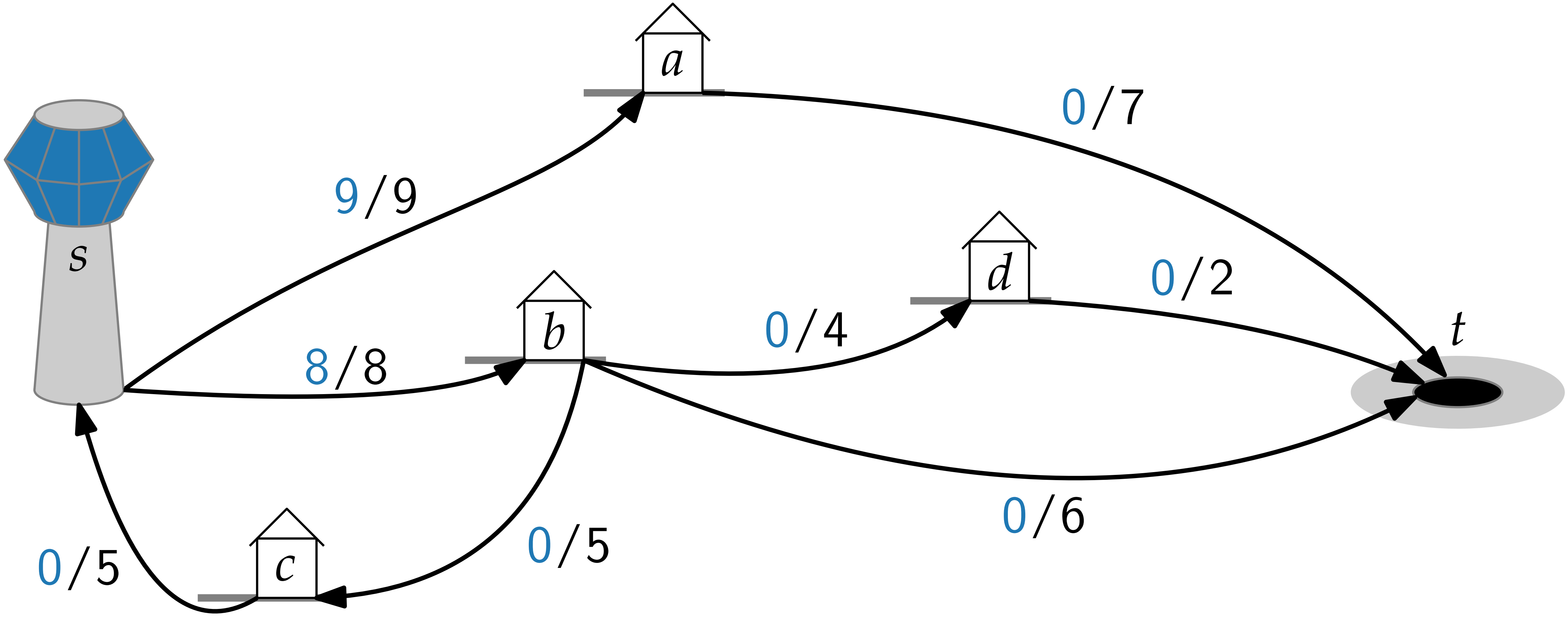
for the next phase. Our algorithm abandons the idea of finding a flow in each phase and also abandons the idea of global phases. Instead, our algorithm maintains a preflow in the original network and pushes local flow excess toward the sink along what it estimates to be shortest paths in the residual graph. This pushing of flow changes the residual graph, and paths to the sink may become saturated. Excess that cannot be moved to the sink is returned to the source, also along estimated shortest paths. Only when the algorithm terminates does the preflow become a flow, and then it is a maximum flow.

The Push–Relabel Idea

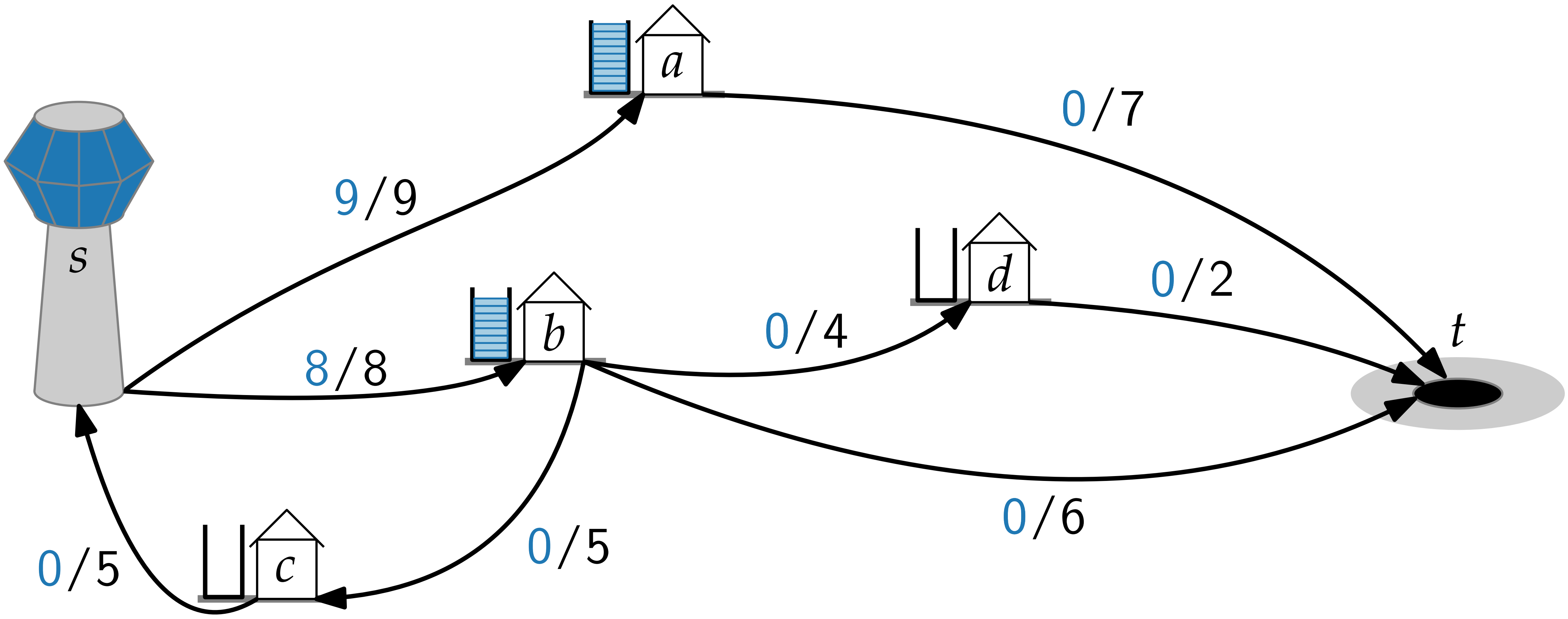
The Push–Relabel Idea



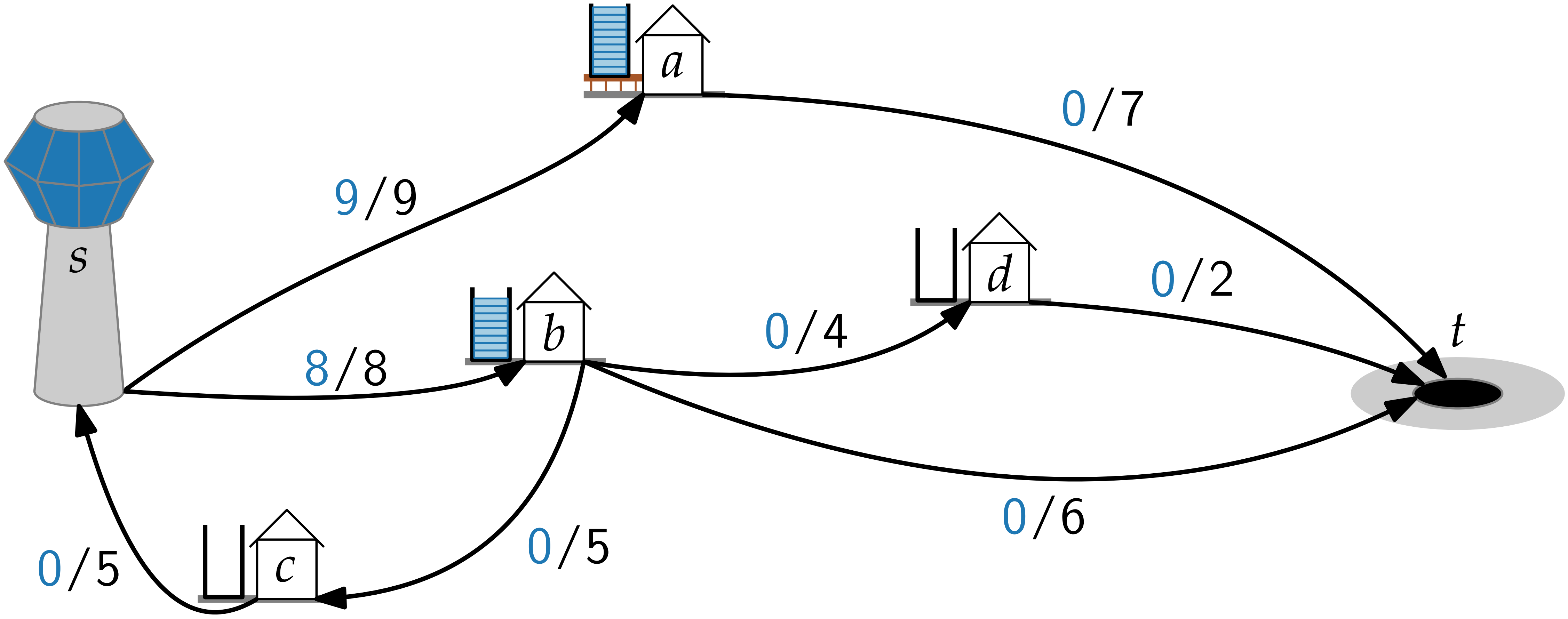
The Push–Relabel Idea



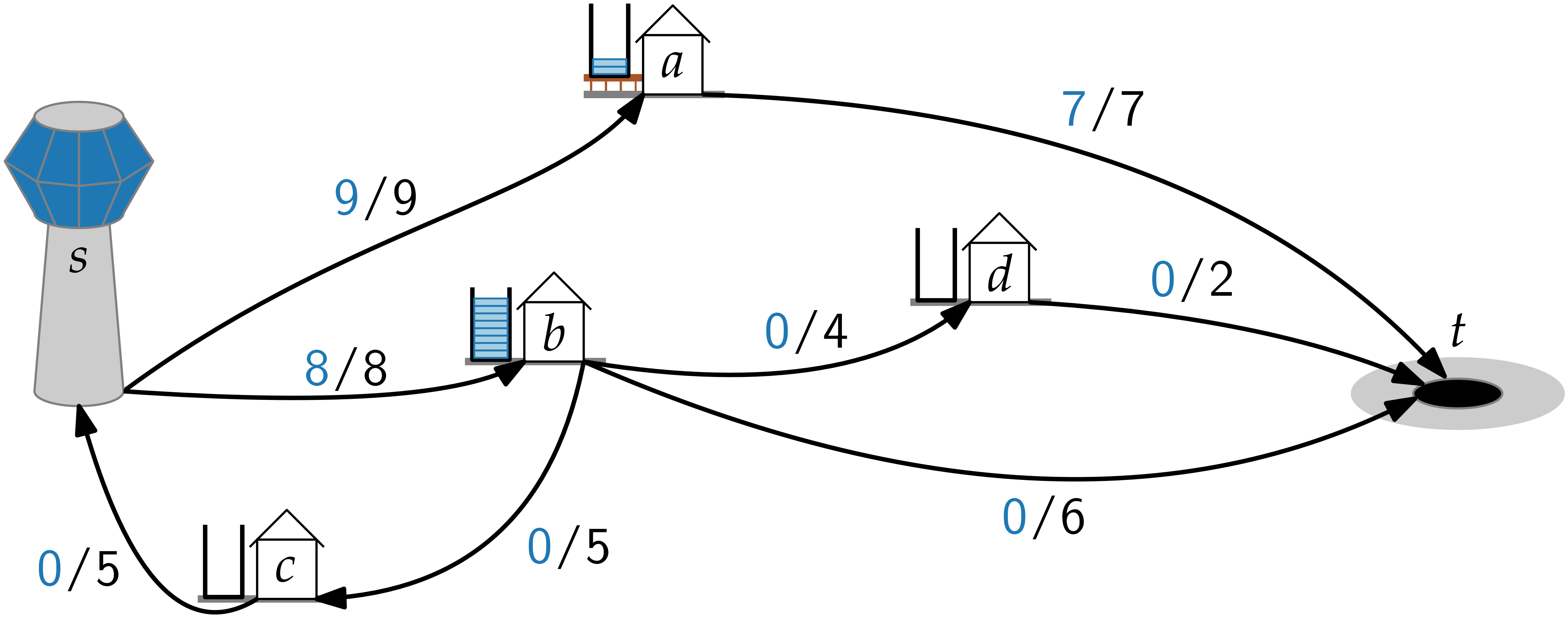
The Push–Relabel Idea



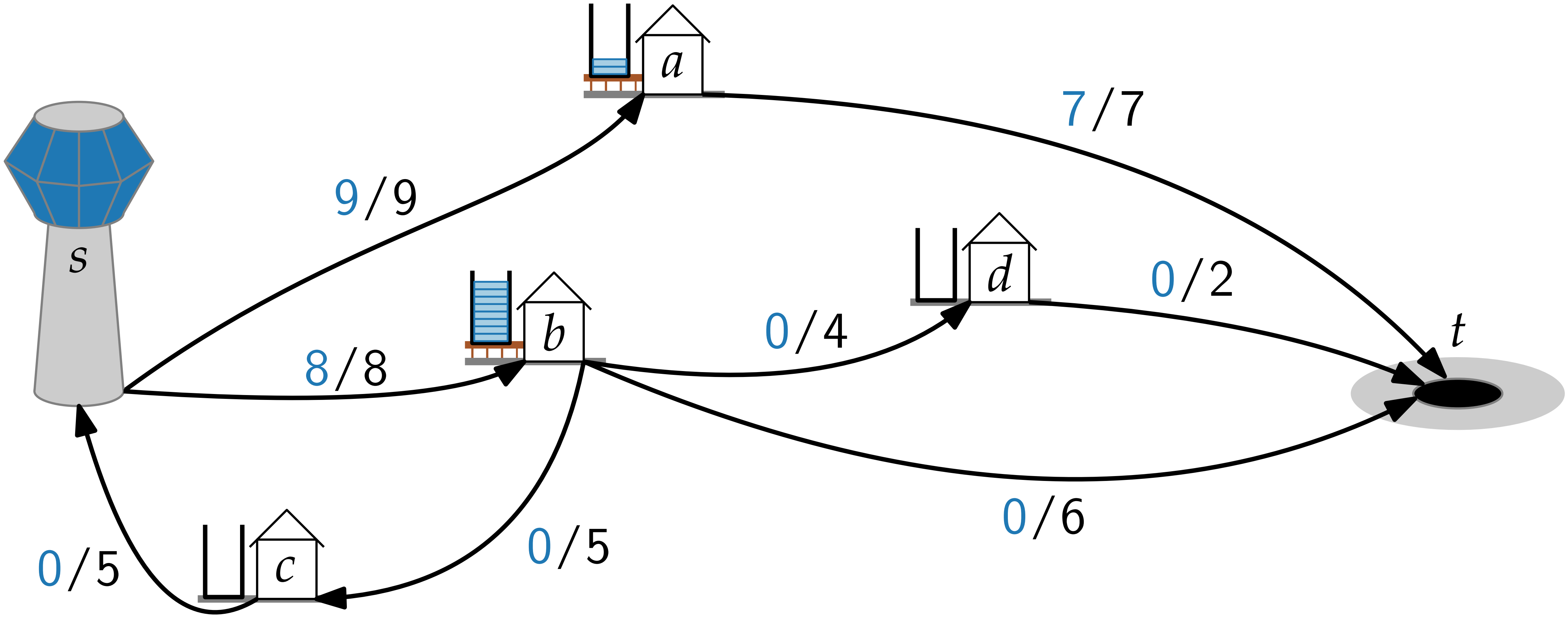
The Push–Relabel Idea



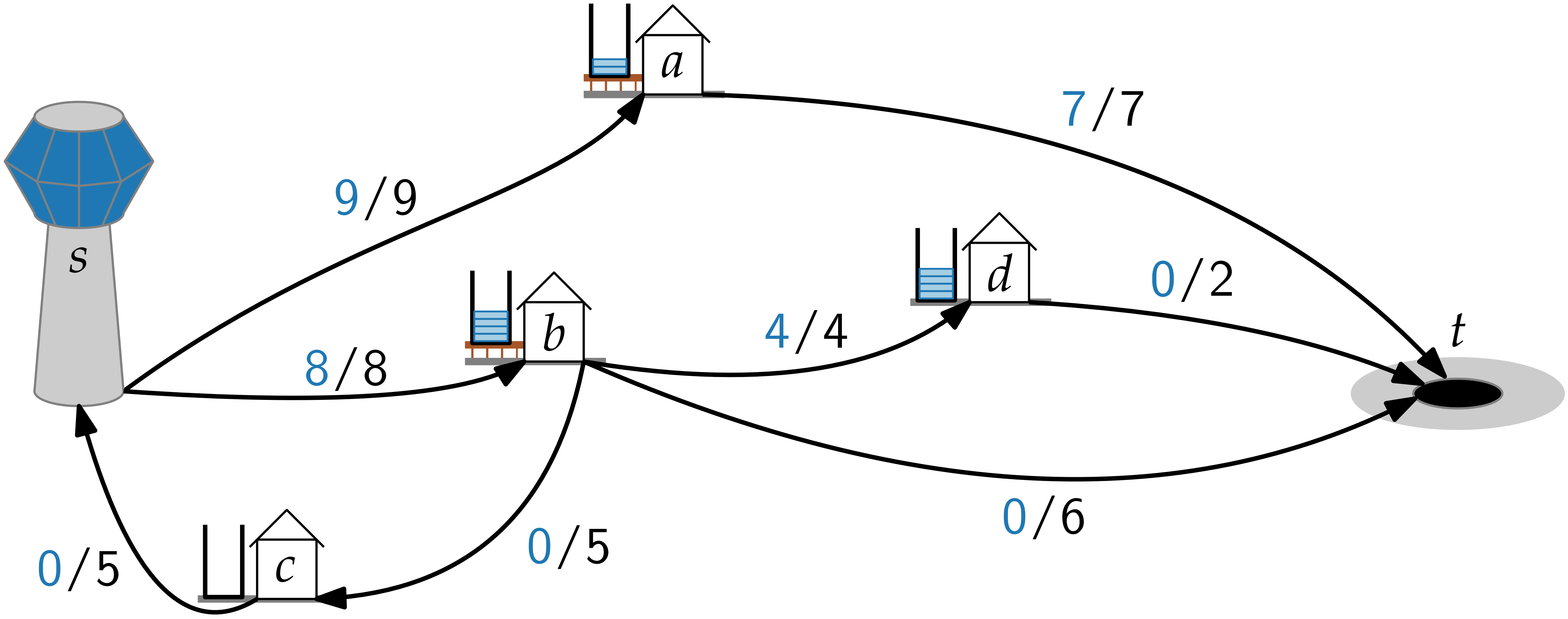
The Push–Relabel Idea



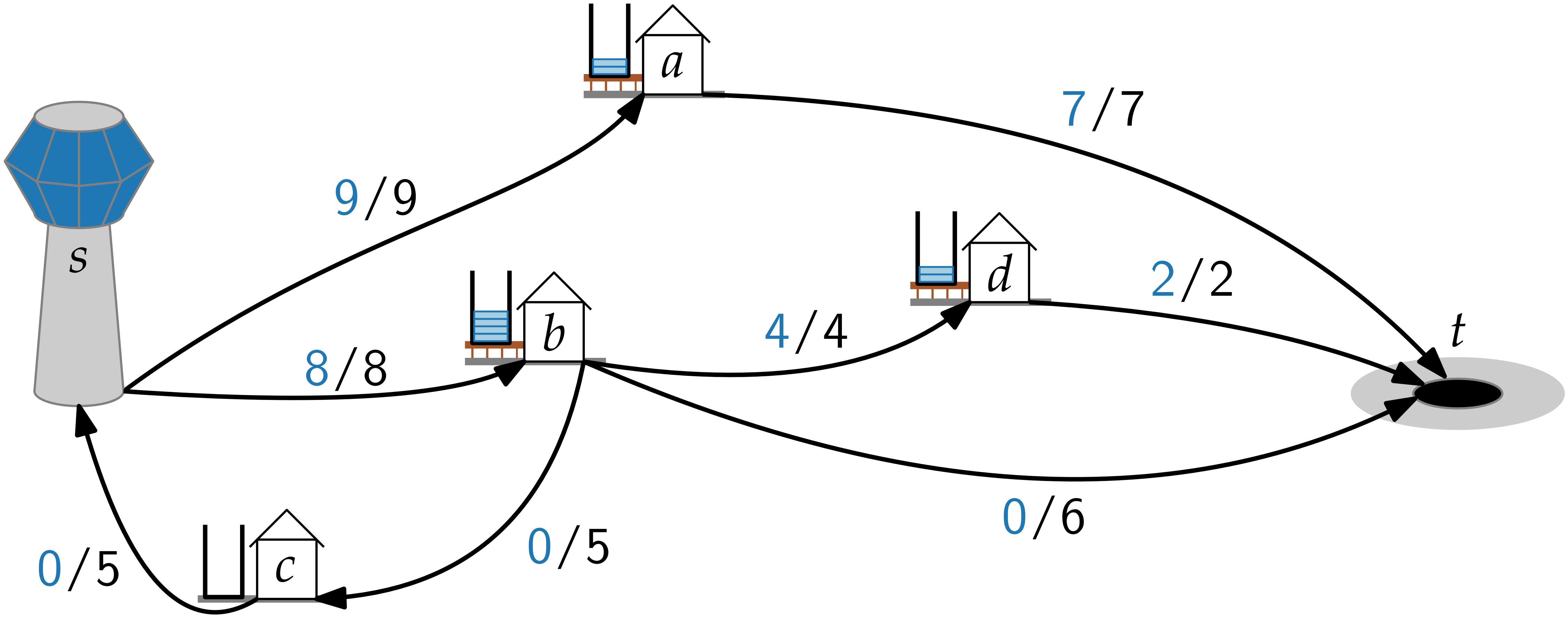
The Push–Relabel Idea



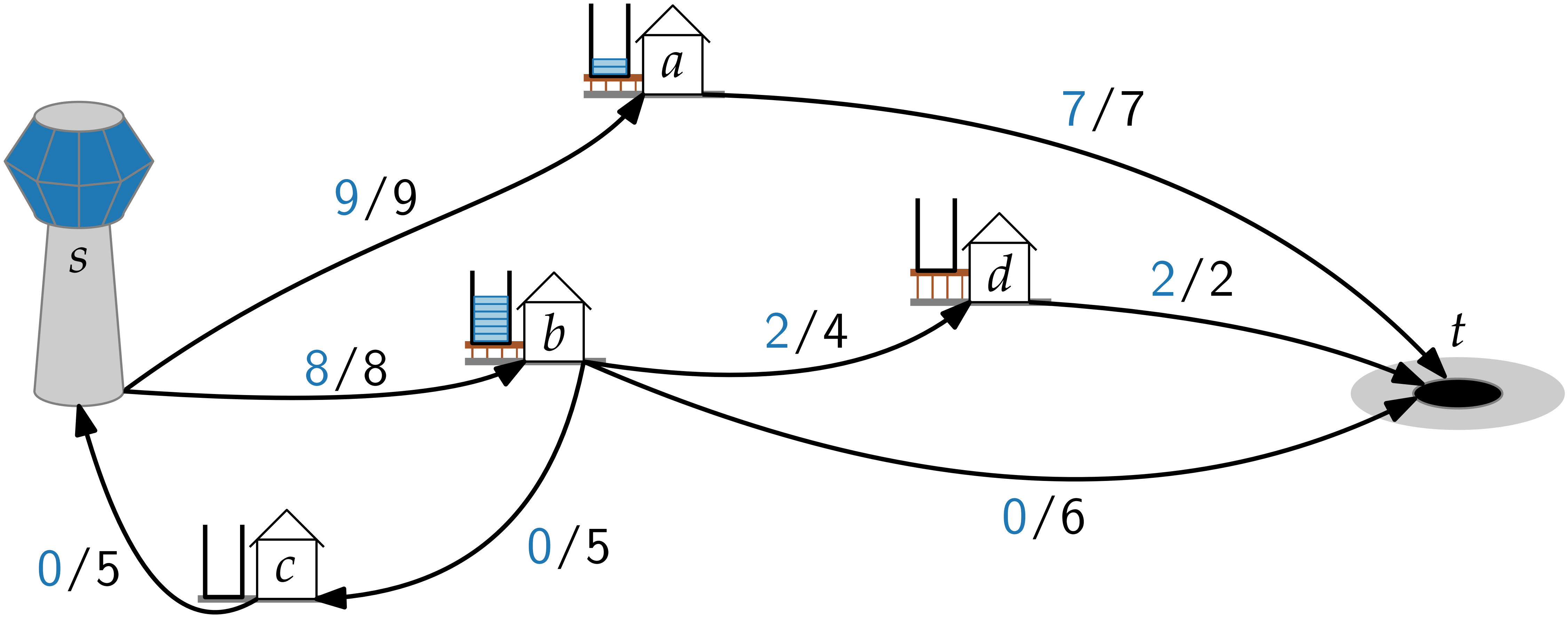
The Push–Relabel Idea



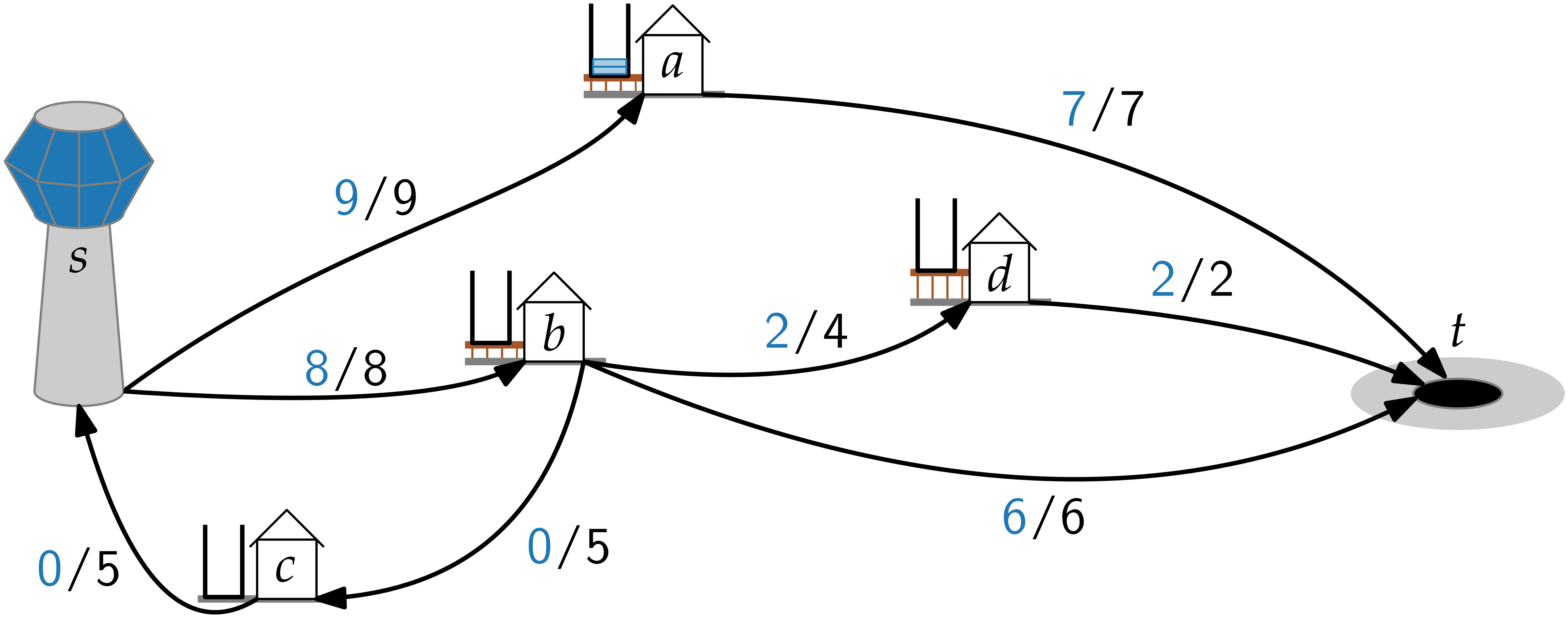
The Push–Relabel Idea



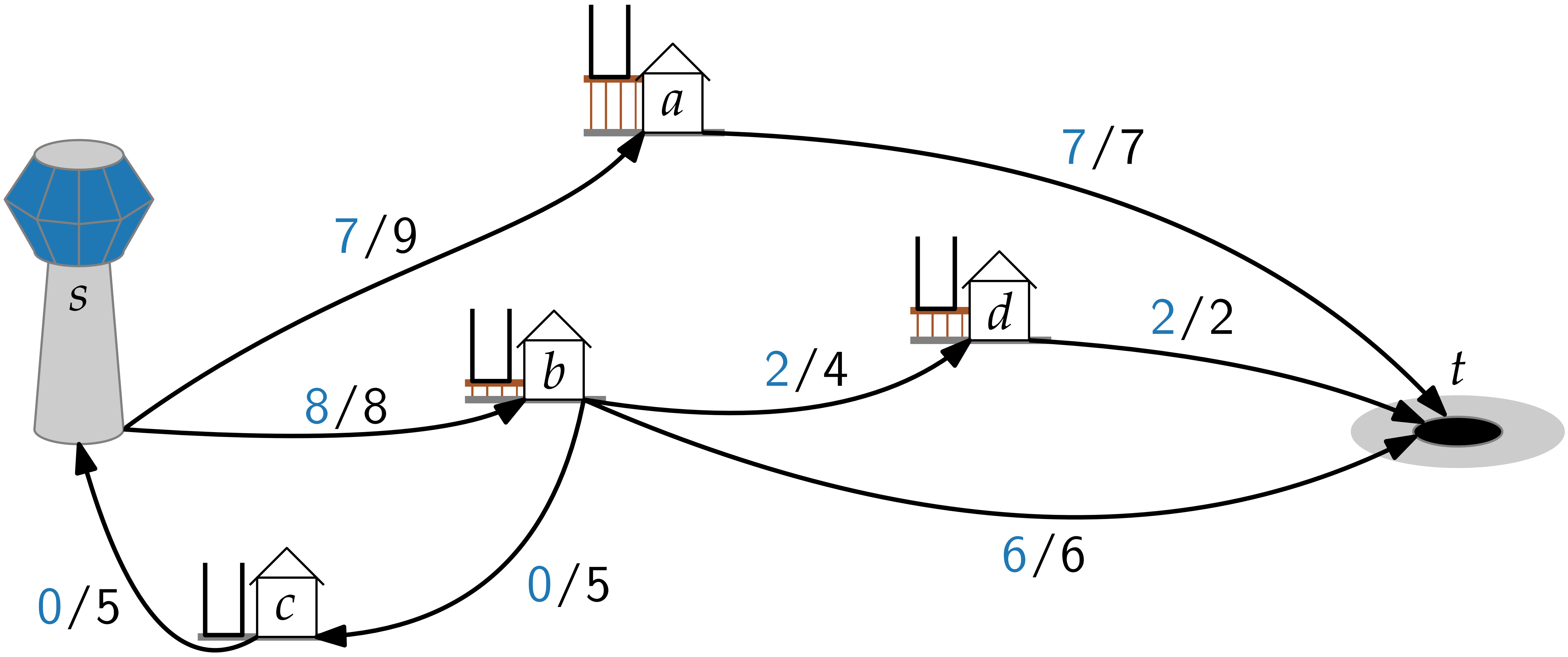
The Push–Relabel Idea



The Push–Relabel Idea



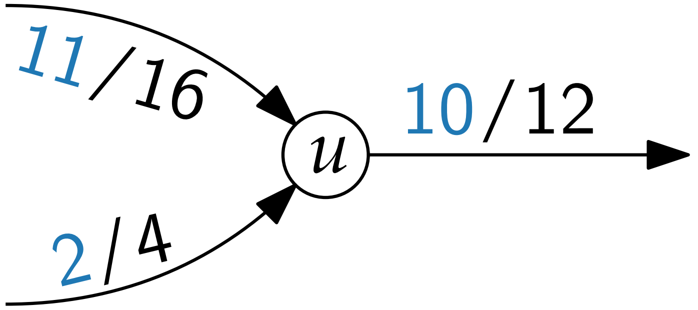
The Push–Relabel Idea



Preflow, Excess Flow, and Height

A **preflow** in G is a real-value function $f\colon E(G) \to \mathbb{R}$ that satisfies the capacity constraints and, for each $u \in V(G) \setminus \{s\}$,

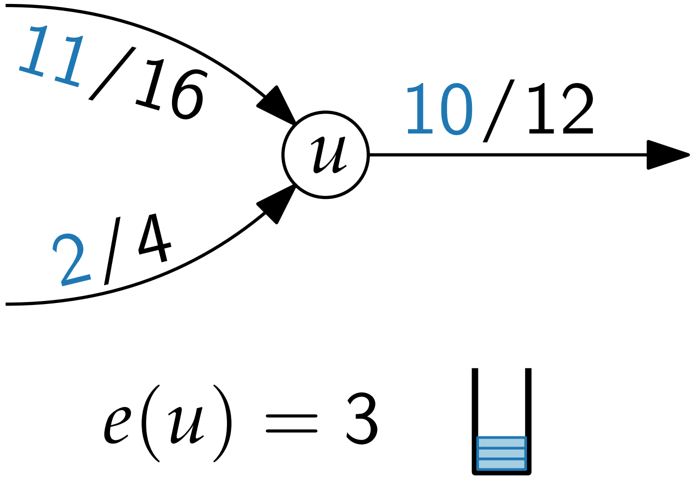
- $\sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v) \geq 0.$



Preflow, Excess Flow, and Height

A **preflow** in G is a real-value function $f\colon E(G) \to \mathbb{R}$ that satisfies the capacity constraints and, for each $u \in V(G) \setminus \{s\}$,

- $\sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v) \geq 0.$



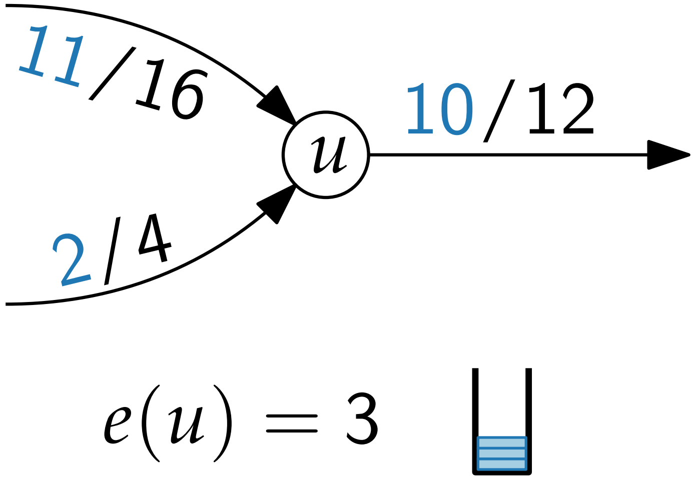
The **excess flow** of a vertex u is

- $e(u) = \sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v).$

Preflow, Excess Flow, and Height

A **preflow** in G is a real-value function $f\colon E(G) \to \mathbb{R}$ that satisfies the capacity constraints and, for each $u \in V(G) \setminus \{s\}$,

- $\sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v) \geq 0.$



The **excess flow** of a vertex u is

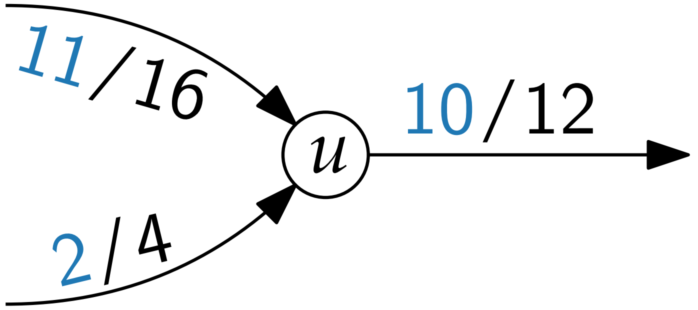
- $e(u) = \sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v).$

A vertex u is called **overflowing** when $e(u) > 0$.

Preflow, Excess Flow, and Height

A **preflow** in G is a real-value function $f\colon E(G) \to \mathbb{R}$ that satisfies the capacity constraints and, for each $u \in V(G) \setminus \{s\}$,

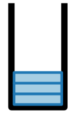
- $\sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v) \geq 0.$



The **excess flow** of a vertex u is

- $e(u) = \sum_{v \in N^-(u)} f(v,u) - \sum_{v \in N^+(u)} f(u,v).$

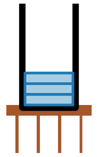
$e(u) = 3$



A vertex u is called **overflowing** when $e(u) > 0$.

For a flow network G with preflow f, a **height function** is a function $h\colon V(G) \to \mathbb{N}$ such that

- $h(s) = |V(G)|$,
- $h(t) = 0$, and
- $h(u) \leq h(v) + 1$ for every residual edge $(u,v) \in E(G_f)$.



The PUSH Operation

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u, v))$ overflow from u to v.

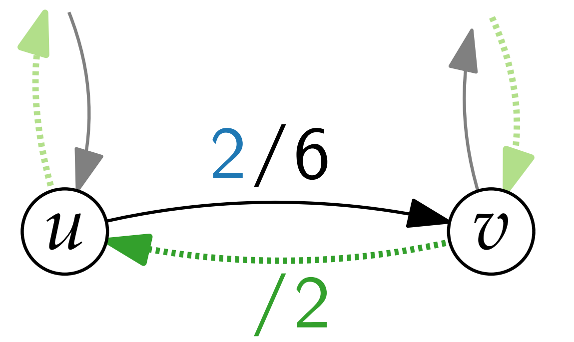
PUSH(u, v)

The Push Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

Push(u, v)

Example.



$e(u) = 5$ $e(v) = 1$
$h(u) = 4$ $h(v) = 3$

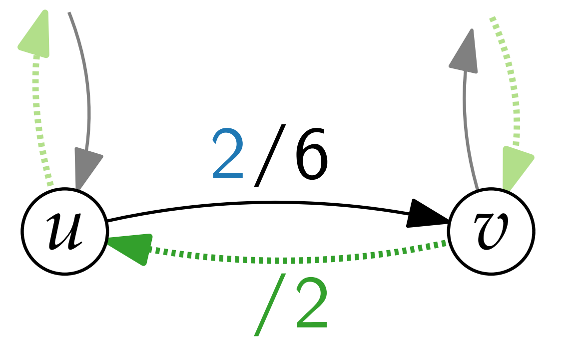
The PUSH Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

PUSH(u,v)

$\Delta = \min(e(u), c_f(u,v))$

Example.



$e(u) = 5$ $e(v) = 1$
$h(u) = 4$ $h(v) = 3$

PUSH(u,v)

$\Delta =$

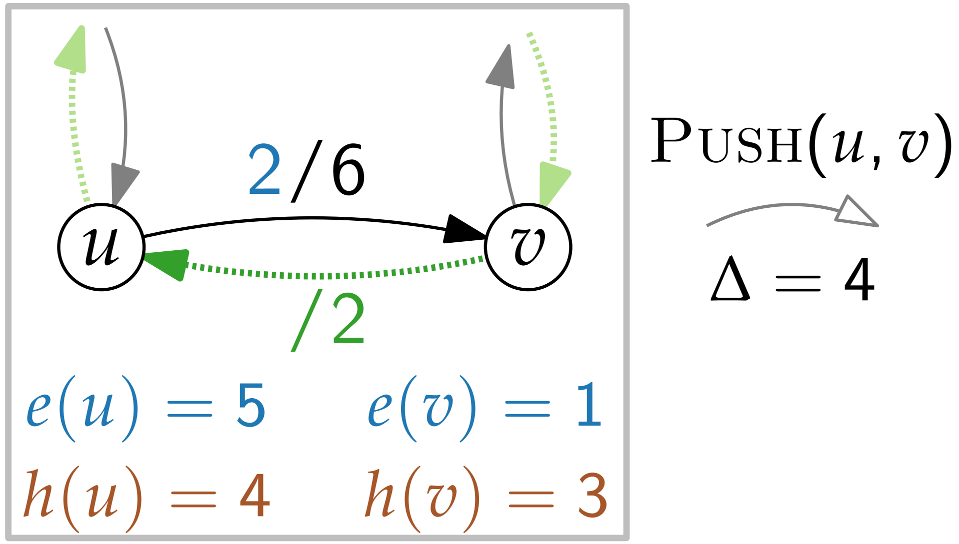
The Push Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

Push(u, v)

$\Delta = \min(e(u), c_f(u,v))$

Example.



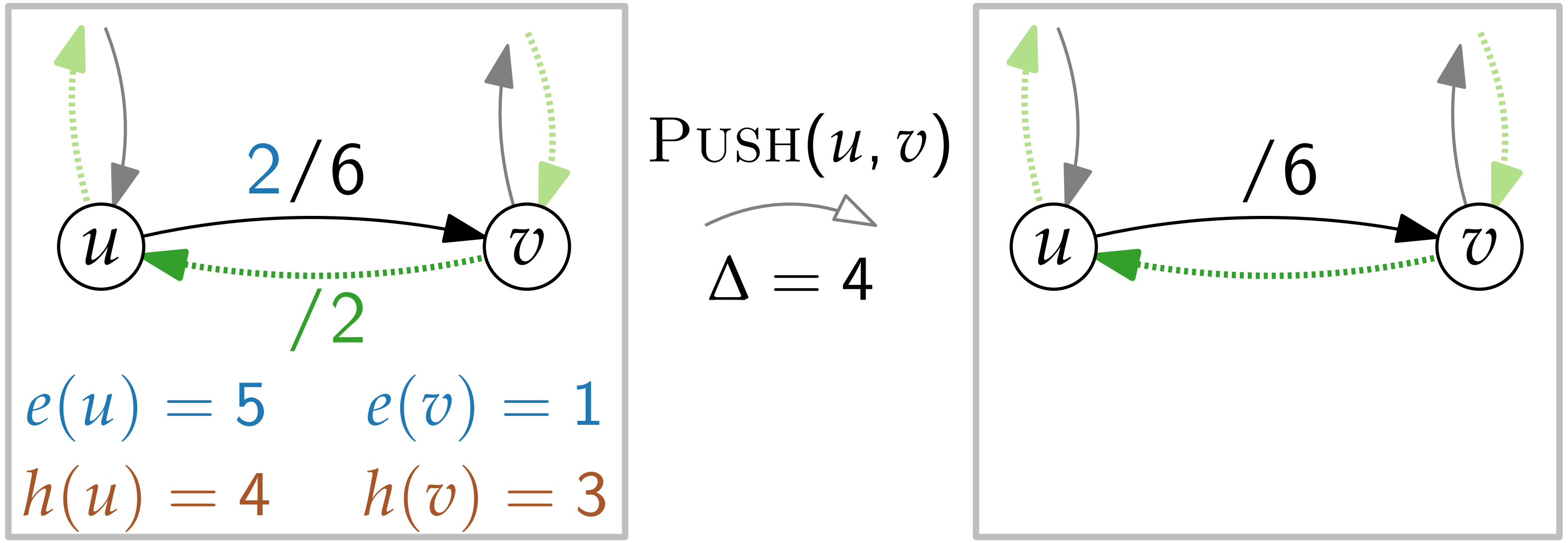
The Push Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

Push(u, v)

$\Delta = \min(e(u), c_f(u,v))$

Example.

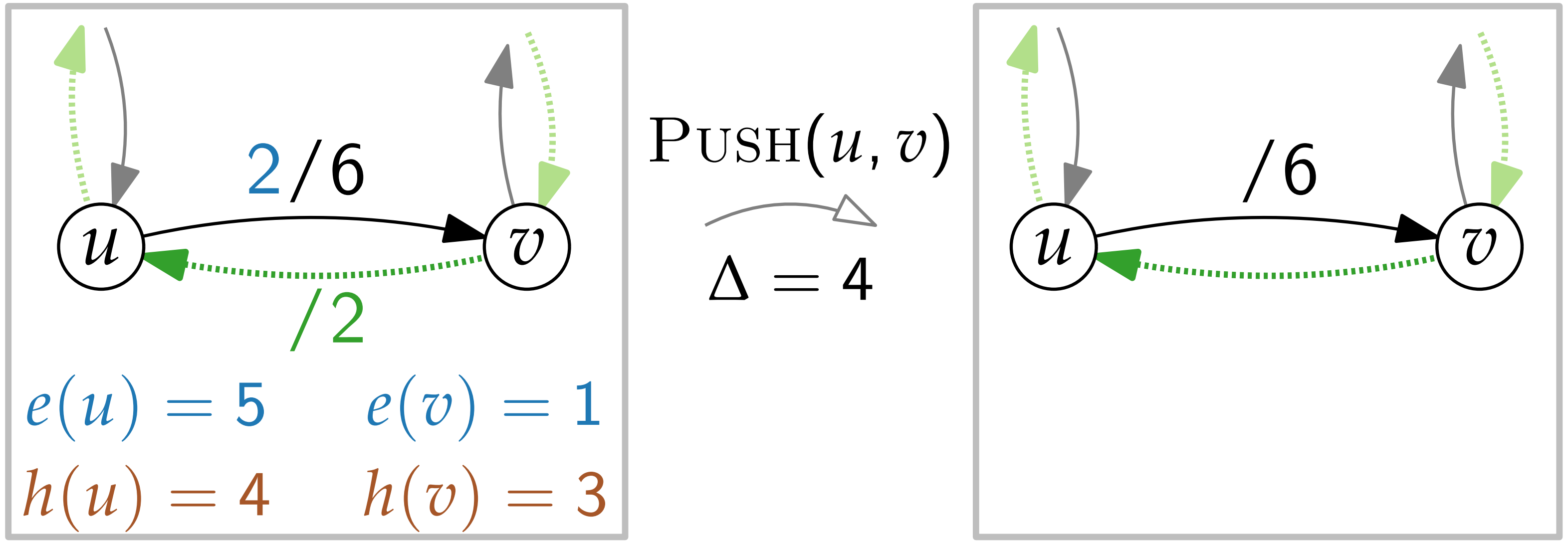


The Push Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

Push(u,v)
$\Delta = \min(e(u), c_f(u,v))$
if $(u,v) \in E(G)$ **then**
  $f(u,v) = f(u,v) + \Delta$
else
  $f(v,u) = f(v,u) - \Delta$

Example.



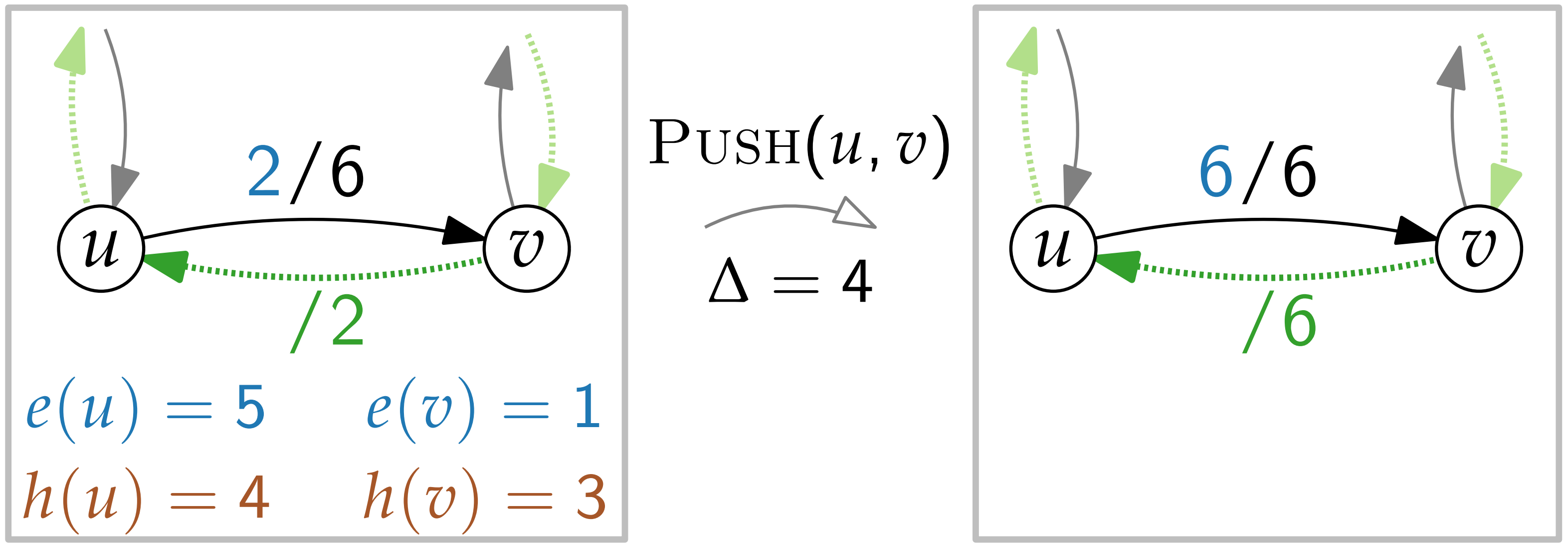
The Push Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

Push(u, v)

  $\Delta = \min(e(u), c_f(u,v))$
  if $(u,v) \in E(G)$ **then**
    $f(u,v) = f(u,v) + \Delta$
  else
    $f(v,u) = f(v,u) - \Delta$

Example.

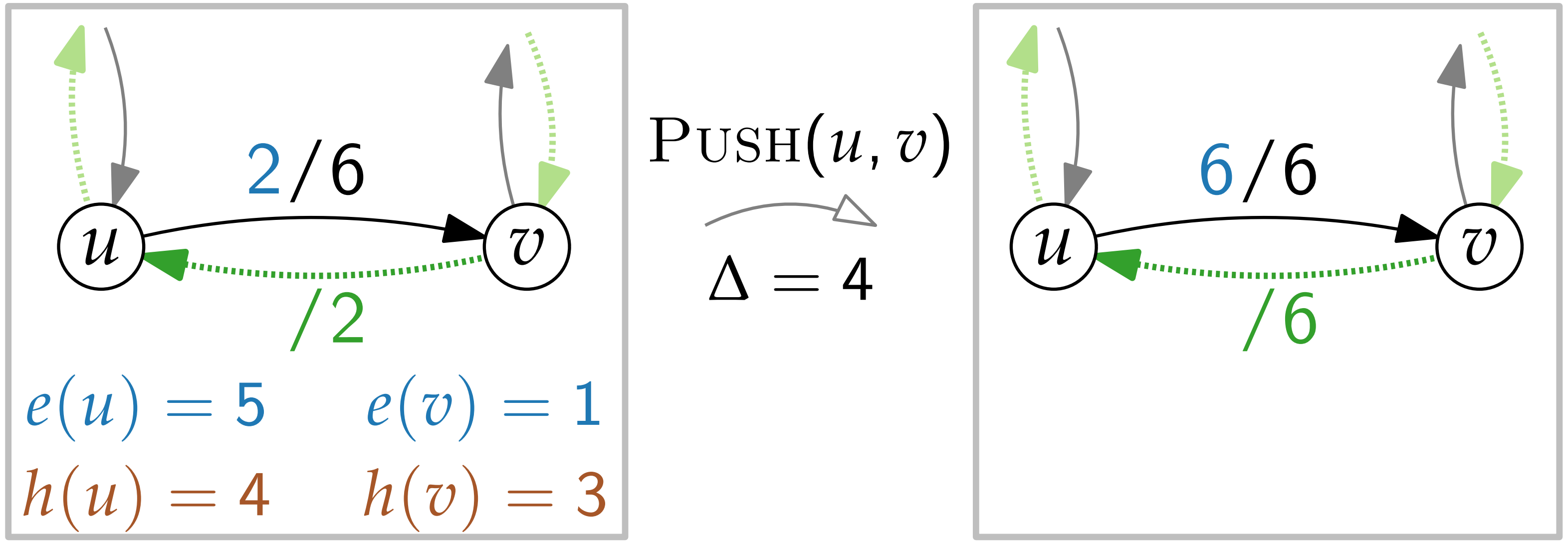


The PUSH Operation

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u,v))$ overflow from u to v.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |  f(u, v) = f(u, v) + Δ
  else
  |_ f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Example.

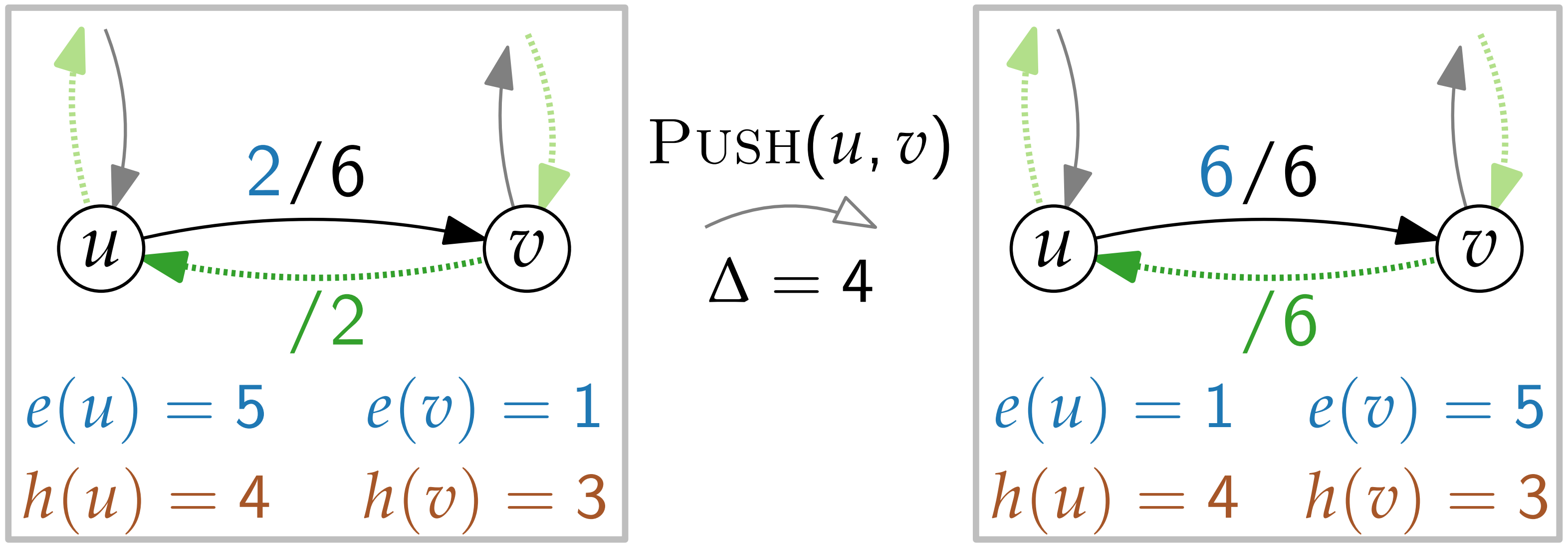


The PUSH Operation

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.
Effect: Push $\min(e(u), c_f(u, v))$ overflow from u to v.

PUSH(u, v)
- $\Delta = \min(e(u), c_f(u, v))$
- **if** $(u, v) \in E(G)$ **then**
  - $f(u, v) = f(u, v) + \Delta$
- **else**
  - $f(v, u) = f(v, u) - \Delta$
- $e(u) = e(u) - \Delta$
- $e(v) = e(v) + \Delta$

Example.



The Relabel Operation

Condition: u is overflowing and $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Effect: Increase the height of u.

Relabel(u)

$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$

The RELABEL Operation

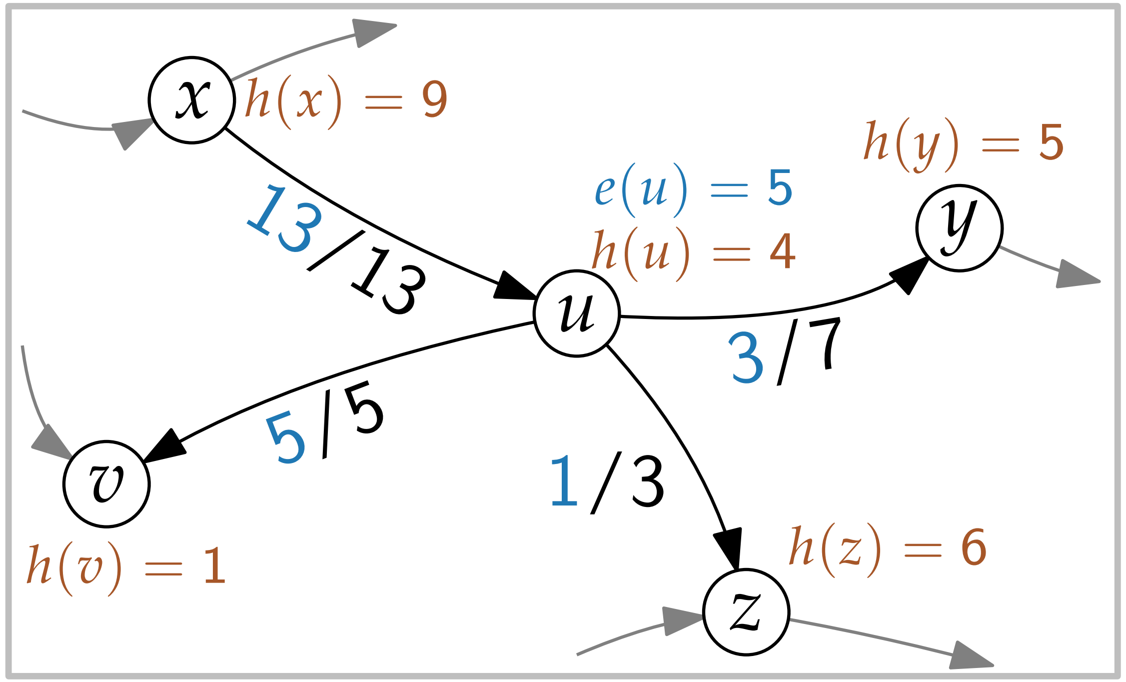
Condition: u is overflowing and $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Effect: Increase the height of u.

RELABEL(u)

$$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$$

Example.



The Relabel Operation

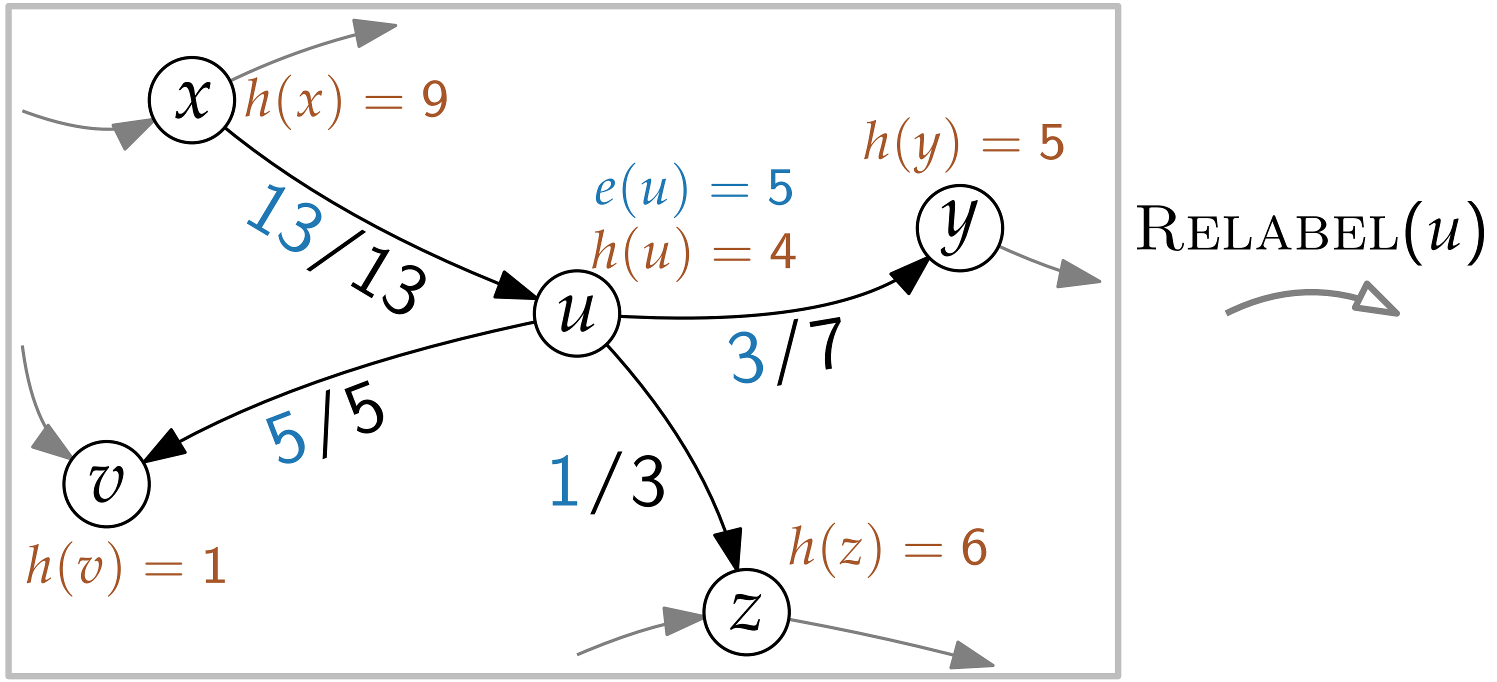
Condition: u is overflowing and $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Effect: Increase the height of u.

Relabel(u)

$$h(u) = 1 + \min\{h(v) \colon (u,v) \in E(G_f)\}$$

Example.



The RELABEL Operation

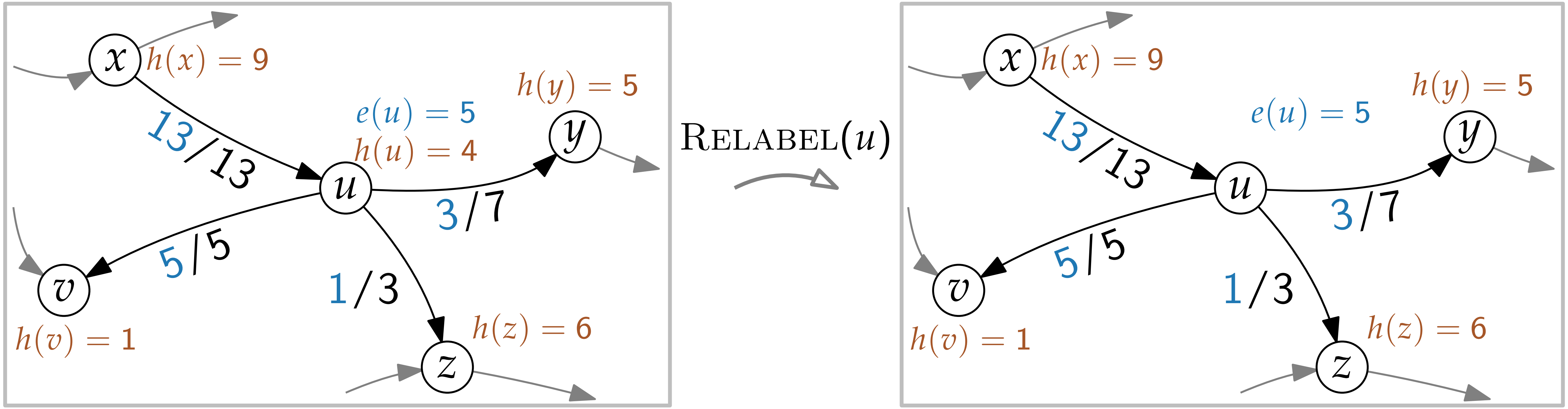
Condition: u is overflowing and $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Effect: Increase the height of u.

RELABEL(u)

$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$

Example.



The RELABEL Operation

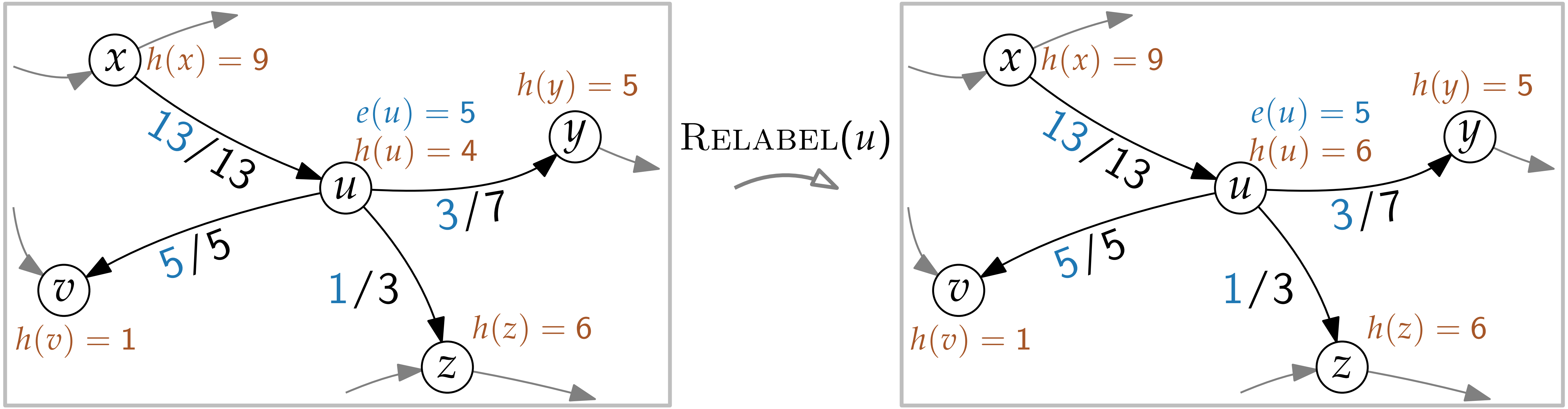
Condition: u is overflowing and $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Effect: Increase the height of u.

RELABEL(u)

$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$

Example.



The Push–Relabel Algorithm

Push–Relabel(G, c, s, t)

InitPreflow(G, c, s)
while $\exists$ applicable Push or Relabel operation X **do**
  apply X

The Push–Relabel Algorithm

Push–Relabel(G, c, s, t)

InitPreflow(G, c, s)
while $\exists$ applicable Push or Relabel operation X **do**
  apply X

InitPreflow(G, c, s)

foreach $v \in V(G)$ **do** $h(v) = 0$; $e(v) = 0$
$h(s) = |V(G)|$
foreach $(u, v) \in E(G)$ **do** $f(u, v) = 0$
foreach $v \in N^+(s)$ **do**
  $f(s, v) = c(s, v)$
  $e(v) = c(s, v)$

The Push–Relabel Algorithm

Push–Relabel(G, c, s, t)

InitPreflow(G, c, s)
while $\exists$ applicable Push or Relabel operation X **do**
  apply X

InitPreflow(G, c, s)

foreach $v \in V(G)$ **do** $h(v) = 0$; $e(v) = 0$
$h(s) = |V(G)|$
foreach $(u, v) \in E(G)$ **do** $f(u, v) = 0$
foreach $v \in N^+(s)$ **do**
  $f(s, v) = c(s, v)$
  $e(v) = c(s, v)$

- initializes heights
- pushes max flow over every edge that leaves s

Correctness

Part 1.

If the algorithm terminates, the preflow is a maximum flow.

- The algorithm maintains f as a preflow and h as a height function.
- If an overflowing vertex exists, the algorithm can continue.
- The sink t is not reachable from source s in G_f.

Correctness

Part 1.

If the algorithm terminates, the preflow is a maximum flow.

- The algorithm maintains f as a preflow and h as a height function.
- If an overflowing vertex exists, the algorithm can continue.
- The sink t is not reachable from source s in G_f.

Part 2.

The algorithm terminates and the heights stay finite.

- Find upper bound on heights.
- Find upper bound for the number of calls to RELABEL.
- Find upper bound for the number of calls to PUSH.

Correctness

Part 1.

If the algorithm terminates, the preflow is a maximum flow.

- The algorithm maintains f as a preflow and h as a height function.
- If an overflowing vertex exists, the algorithm can continue.
- The sink t is not reachable from source s in G_f.

Part 2.

The algorithm terminates and the heights stay finite.

- Find upper bound on heights.
- Find upper bound for the number of calls to Relabel.
- Find upper bound for the number of calls to Push.

Maintaining the Preflow

Lemma 1.
The push–relabel algorithm maintains a preflow f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$

$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Preflow

Lemma 1.
The push–relabel algorithm maintains a preflow f.

Proof.

- InitPreflow initializes a preflow f. ✓

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

Push(u, v)
 $\Delta = \min(e(u), c_f(u, v))$
 if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
 else
  $f(v, u) = f(v, u) - \Delta$
 $e(u) = e(u) - \Delta$
 $e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)
 $h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Preflow

Lemma 1.
The push–relabel algorithm maintains a preflow f.

Proof.

- $\textsc{InitPreflow}$ initializes a preflow f. ✓
- $\textsc{Relabel}(u)$ does not affect f. ✓

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

$\textsc{Push}(u, v)$

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

$\textsc{Relabel}(u)$

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Preflow

Lemma 1.
The push–relabel algorithm maintains a preflow f.

Proof.

- InitPreflow initializes a preflow f. ✓
- Relabel(u) does not affect f. ✓
- Push(u, v) maintains f as a preflow. ✓

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

```
Relabel(u)
  h(u) = 1 + min{h(v) : (u, v) ∈ E(G_f)}
```

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

Push(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓
- Under Push(u, v), h remains a height function:

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

Push(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓
- Under Push(u, v), h remains a height function:
  - If (v, u) is added to $E(G_f)$, then $h(v) = h(u) - 1 < h(u) + 1$.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |_  f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

```
Relabel(u)
  h(u) = 1 + min{h(v): (u, v) ∈ E(G_f)}
```

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓
- Under Push(u, v), h remains a height function:
  - If (v, u) is added to $E(G_f)$, then $h(v) = h(u) - 1 < h(u) + 1$. ✓
  - If (u, v) is removed from E_f, then ✓.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

Push(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓
- Under Push(u, v), h remains a height function:
  - If (v, u) is added to $E(G_f)$, then $h(v) = h(u) - 1 < h(u) + 1$. ✓
  - If (u, v) is removed from E_f, then ✓.
- Under Relabel(u), h remains a height function:
  - $(u, v) \in E_f$, then $h(u) \le h(v) + 1$
  - $(w, u) \in E_f$, then $h(w) < h(u) + 1$

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \le h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |  f(u, v) = f(u, v) + Δ
  else
  └  f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Condition: u is overflowing and $h(u) \le h(v)\ \forall v \in N_f^+(u)$.

```
Relabel(u)
  h(u) = 1 + min{h(v): (u, v) ∈ E(G_f)}
```

Maintaining the Height Function

Lemma 2.
The push–relabel algorithm maintains h as a height function.

Proof.

- InitPreflow initializes h as a height function. ✓
- Under Push(u, v), h remains a height function:
  - If (v, u) is added to $E(G_f)$, then $h(v) = h(u) - 1 < h(u) + 1$. ✓
  - If (u, v) is removed from E_f, then ✓.
- Under Relabel(u), h remains a height function: ✓
  - $(u, v) \in E_f$, then $h(u) \leq h(v) + 1$
  - $(w, u) \in E_f$, then $h(w) < h(u) + 1$

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

Push(u, v)

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Continuation

Lemma 3.
If a vertex u is overflowing, either a $\textsc{Push}$ or a $\textsc{Relabel}$ operation applies to u.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

$\textsc{Push}(u, v)$

$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v) \; \forall v \in N_f^+(u)$.

$\textsc{Relabel}(u)$

$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Continuation

Lemma 3.
If a vertex u is overflowing, either a PUSH or a RELABEL operation applies to u.

Proof.
By Lemma 2, h is a height function. Thus, we have

- $h(u) \leq h(v) + 1$ for every $v \in N_f^+(u)$.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u, v)
- $\Delta = \min(e(u), c_f(u, v))$
- **if** $(u, v) \in E(G)$ **then**
  - $f(u, v) = f(u, v) + \Delta$
- **else**
  - $f(v, u) = f(v, u) - \Delta$
- $e(u) = e(u) - \Delta$
- $e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)
- $h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Continuation

Lemma 3.
If a vertex u is overflowing, either a $\textsc{Push}$ or a $\textsc{Relabel}$ operation applies to u.

Proof.
By Lemma 2, h is a height function. Thus, we have

- $h(u) \leq h(v) + 1$ for every $v \in N_f^+(u)$.

If no $\textsc{Push}$ operation is valid for $(u, v) \in E_f$, then

- $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

$\textsc{Push}(u, v)$
$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
$f(u, v) = f(u, v) + \Delta$
else
$f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

$\textsc{Relabel}(u)$
$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$

Continuation

Lemma 3.
If a vertex u is overflowing, either a $\textsc{Push}$ or a $\textsc{Relabel}$ operation applies to u.

Proof.
By Lemma 2, h is a height function. Thus, we have

- $h(u) \leq h(v) + 1$ for every $v \in N_f^+(u)$.

If no $\textsc{Push}$ operation is valid for $(u, v) \in E_f$, then

- $h(u) \leq h(v)$ for every $v \in N_f^+(u)$.

Therefore, $\textsc{Relabel}(u)$ is applicable.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

```
Relabel(u)
  h(u) = 1 + min{h(v): (u, v) ∈ E(G_f)}
```

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Proof.
Suppose that there is a *simple* path $s = v_0, v_1, \ldots, v_k = t$ in G_f.
Then

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Proof.

Suppose that there is a *simple* path $s = v_0, v_1, \ldots, v_k = t$ in G_f.
Then

- $(v_i, v_{i+1}) \in E(G_f)$ for $0 \leq i \leq k - 1$, and

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u,v) \in E(G_f)$

Proof.
Suppose that there is a *simple* path $s = v_0, v_1, \ldots, v_k = t$ in G_f.
Then

- $(v_i, v_{i+1}) \in E(G_f)$ for $0 \leq i \leq k-1$, and
- $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k-1$.

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Proof.

Suppose that there is a *simple* path $s = v_0, v_1, \ldots, v_k = t$ in G_f.
Then

- $(v_i, v_{i+1}) \in E(G_f)$ for $0 \leq i \leq k-1$, and
- $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k-1$.

$$\Rightarrow h(s) \leq h(t) + k = k$$

Reachability of the Sink

Lemma 4.
During the execution of the push–relabel algorithm, there is no path from s to t in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u,v) \in E(G_f)$

Proof.
Suppose that there is a *simple* path $s = v_0, v_1, \ldots, v_k = t$ in G_f.
Then

- $(v_i, v_{i+1}) \in E(G_f)$ for $0 \leq i \leq k-1$, and
- $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k-1$.

$$\Rightarrow h(s) \leq h(t) + k = k$$

But since $k < |V(G)|$, it follows that $h(s) < |V(G)|$. ✗

Correctness of the Algorithm (Part I)

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Correctness of the Algorithm (Part I)

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Proof.

- By Lemma 3, the algorithm stops when there is no overflowing vertex.
- By Lemma 1, f is a preflow.

Correctness of the Algorithm (Part I)

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Proof.

- By Lemma 3, the algorithm stops when there is no overflowing vertex.
- By Lemma 1, f is a preflow.
  $\Rightarrow$ f is a flow.

Correctness of the Algorithm (Part I)

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Proof.

- By Lemma 3, the algorithm stops when there is no overflowing vertex.
- By Lemma 1, f is a preflow.
  $\Rightarrow$ f is a flow.
- By Lemma 2, h is a height function.
- So by Lemma 4, there is no s–t path in G_f.

Correctness of the Algorithm (Part I)

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Proof.

- By Lemma 3, the algorithm stops when there is no overflowing vertex.
- By Lemma 1, f is a preflow.

  $\Rightarrow$ f is a flow.
- By Lemma 2, h is a height function.
- So by Lemma 4, there is no s–t path in G_f.

  $\Rightarrow$ By the Max-Flow Min-Cut Theorem, the flow f is a maximum flow.

Correctness

Part 1. ✓

If the algorithm terminates, the preflow is maximum flow.

- The algorithm maintains f as a preflow and h as a height function.
- If an overflowing vertex exists, the algorithm can continue.
- Sink t is not reachable from source s in G_f.

Correctness

Part 1. ✓

If the algorithm terminates, the preflow is maximum flow.

- The algorithm maintains f as a preflow and h as a height function.
- If an overflowing vertex exists, the algorithm can continue.
- Sink t is not reachable from source s in G_f.

Part 2.

The algorithm terminates and the heights stay finite.

- Find upper bound on heights.
- Find upper bound for the number of calls to Relabel.
- Find upper bound for the number of calls to Push.

Reachability of the Source in the Residual Graph

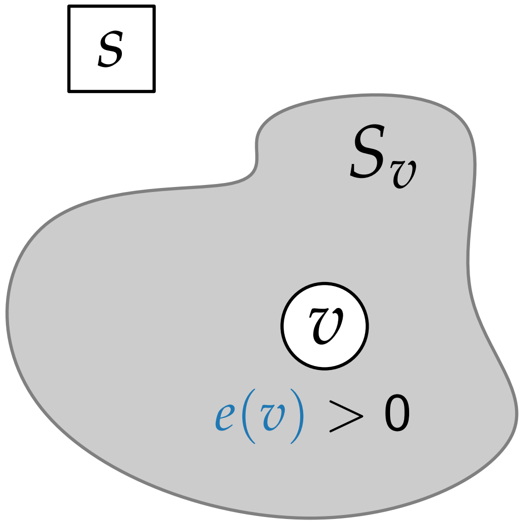
Lemma 6.
For every overflowing vertex v, there is a path from v to s in G_f.

Reachability of the Source in the Residual Graph

Lemma 6.
For every overflowing vertex v,
there is a path from v to s in G_f.

Proof.

- Let S_v be the set of vertices reachable from v in G_f.
- Suppose that $s \notin S_v$.

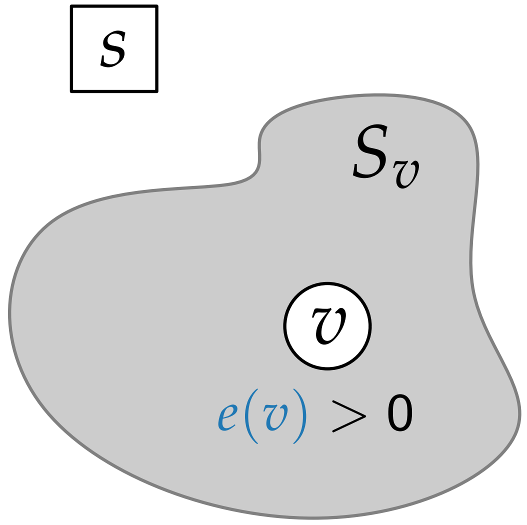


Reachability of the Source in the Residual Graph

Lemma 6.
For every overflowing vertex v,
there is a path from v to s in G_f.

Proof.

- Let S_v be the set of vertices reachable from v in G_f.
- Suppose that $s \notin S_v$.
- Since f is a preflow and $s \notin S_v$, we have $\sum_{w \in S_v} e(w) \geq 0$.

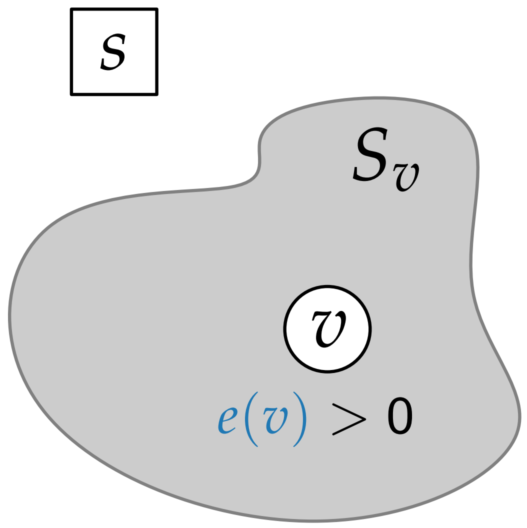


Reachability of the Source in the Residual Graph

Lemma 6.
For every overflowing vertex v, there is a path from v to s in G_f.

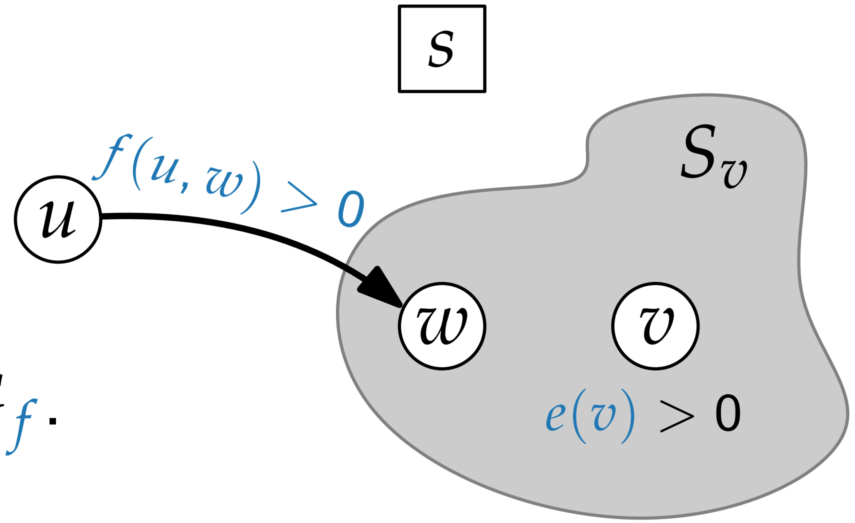
Proof.

- Let S_v be the set of vertices reachable from v in G_f.
- Suppose that $s \notin S_v$.
- Since f is a preflow and $s \notin S_v$, we have $\sum_{w \in S_v} e(w) \geq 0$.
- Since $v \in S_v$, we even have $\sum_{w \in S_v} e(w) > 0$.



Reachability of the Source in the Residual Graph

Lemma 6.
For every overflowing vertex v, there is a path from v to s in G_f.

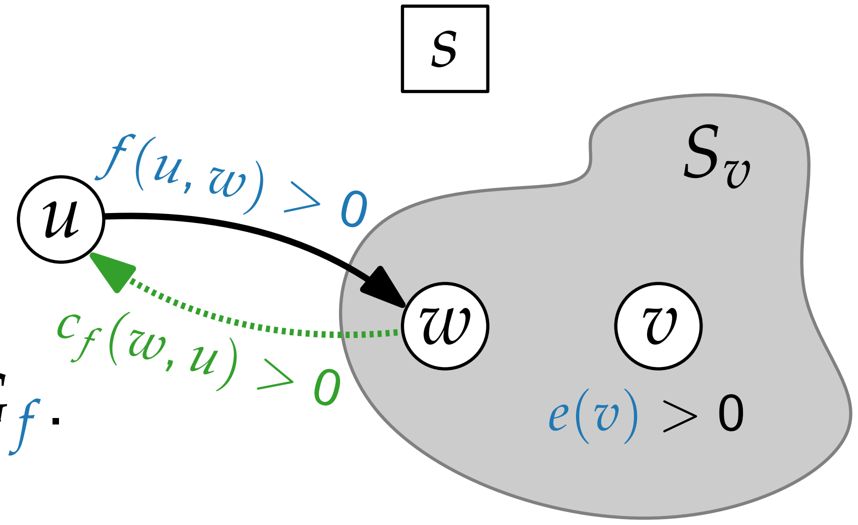


Proof.

- Let S_v be the set of vertices reachable from v in G_f.
- Suppose that $s \notin S_v$.
- Since f is a preflow and $s \notin S_v$, we have $\sum_{w \in S_v} e(w) \geq 0$.
- Since $v \in S_v$, we even have $\sum_{w \in S_v} e(w) > 0$.
- There is an edge (u, w) with $u \notin S_v, w \in S_v$ and $f(u,w) > 0$.

Reachability of the Source in the Residual Graph

Lemma 6.
For every overflowing vertex v,
there is a path from v to s in G_f.



Proof.

- Let S_v be the set of vertices reachable from v in G_f.
- Suppose that $s \notin S_v$.
- Since f is a preflow and $s \notin S_v$, we have $\sum_{w \in S_v} e(w) \geq 0$.
- Since $v \in S_v$, we even have $\sum_{w \in S_v} e(w) > 0$.
- There is an edge (u, w) with $u \notin S_v, w \in S_v$ and $f(u, w) > 0$.
- But then $c_f(w, u) > 0$, meaning u is reachable from v. ✗

Upper Bound on the Height

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)$ $\forall v \in N_f^+(u)$.

Relabel(u)
$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$

Upper Bound on the Height

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Proof.

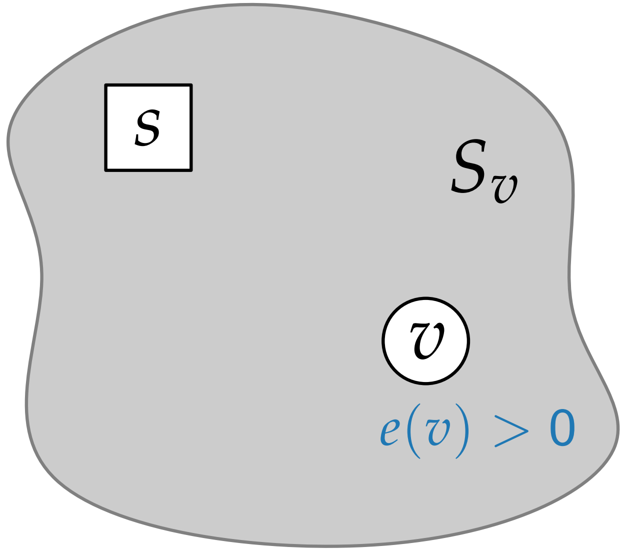
- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)
$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$



Upper Bound on the Height

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Proof.

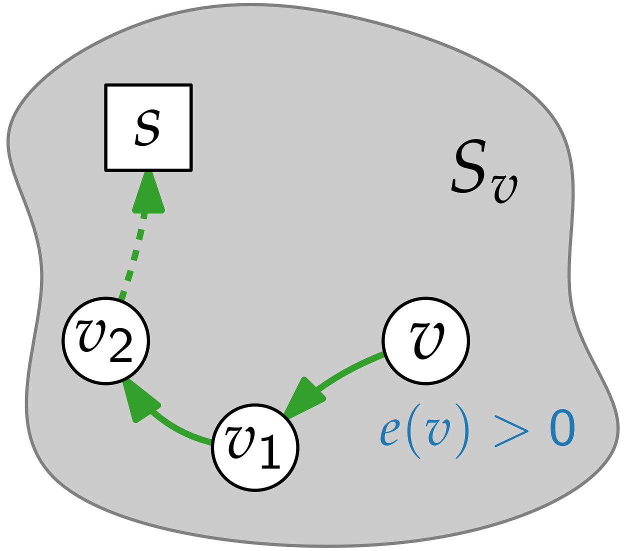
- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.
- By Lemma 6, there is a path $v = v_0, v_1, \ldots, v_k = s$ in G_f.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)
$h(u) = 1 + \min\{h(v) \colon (u, v) \in E(G_f)\}$



Upper Bound on the Height

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Proof.

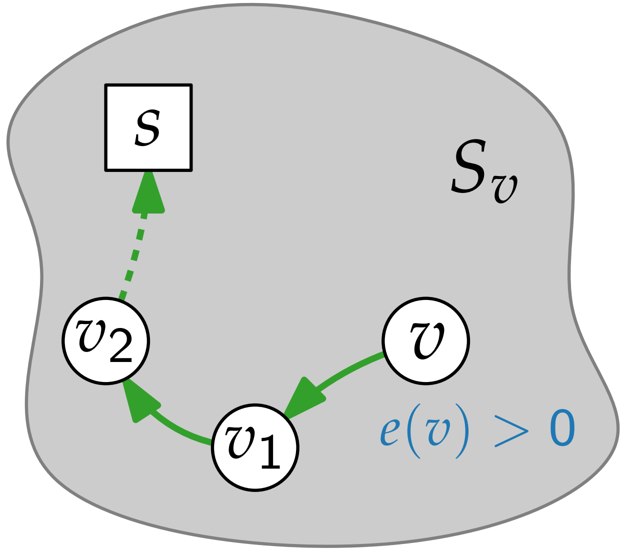
- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.
- By Lemma 6, there is a path $v = v_0, v_1, \ldots, v_k = s$ in G_f.
- Then $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k - 1$.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)
$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$



Upper Bound on the Height

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Height function:

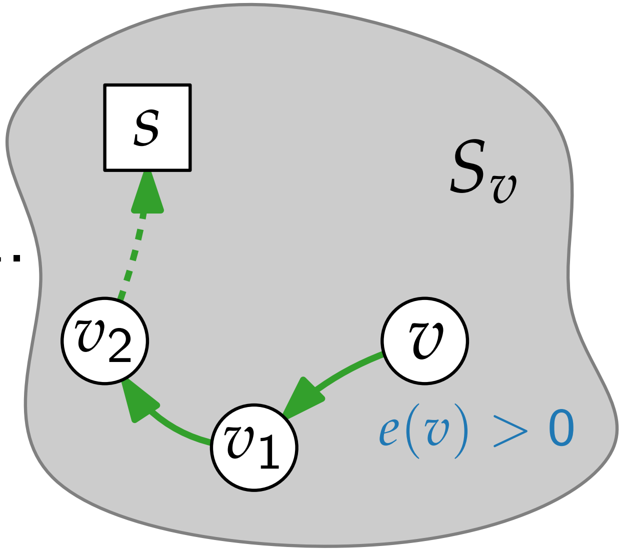
- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u, v) \in E(G_f)$

Condition: u is overflowing and $h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

RELABEL(u)
$h(u) = 1 + \min\{h(v) : (u, v) \in E(G_f)\}$

Proof.

- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.
- By Lemma 6, there is a path $v = v_0, v_1, \ldots, v_k = s$ in G_f.
- Then $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k - 1$.
- Since $k \leq |V(G)| - 1$, we have $h(v) \leq h(s) + k \leq 2|V(G)| - 1$.



Upper Bounds on the Height and # Relabel Operations

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Proof.

- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.
- By Lemma 6, there is a path $v = v_0, v_1, \ldots, v_k = s$ in G_f.
- Then $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k-1$.
- Since $k \leq |V(G)| - 1$, we have $h(v) \leq h(s) + k \leq 2|V(G)| - 1$.

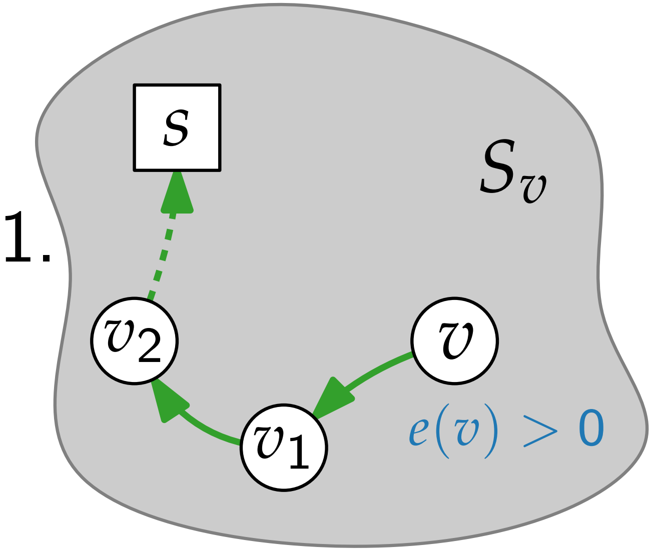
Corollary 8.
The push–relabel algorithm executes
at most Relabel operations.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u,v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)
$h(u) = 1 + \min\{h(v) : (u,v) \in E(G_f)\}$



Upper Bounds on the Height and # Relabel Operations

Lemma 7.
During the push–relabel algorithm,
we have $h(v) \leq 2|V(G)| - 1$ for every $v \in V(G)$.

Proof.

- Statement holds after initialization.
- Let v be an overflowing vertex that is relabeled.
- By Lemma 6, there is a path $v = v_0, v_1, \ldots, v_k = s$ in G_f.
- Then $h(v_i) \leq h(v_{i+1}) + 1$ for $0 \leq i \leq k - 1$.
- Since $k \leq |V(G)| - 1$, we have $h(v) \leq h(s) + k \leq 2|V(G)| - 1$.

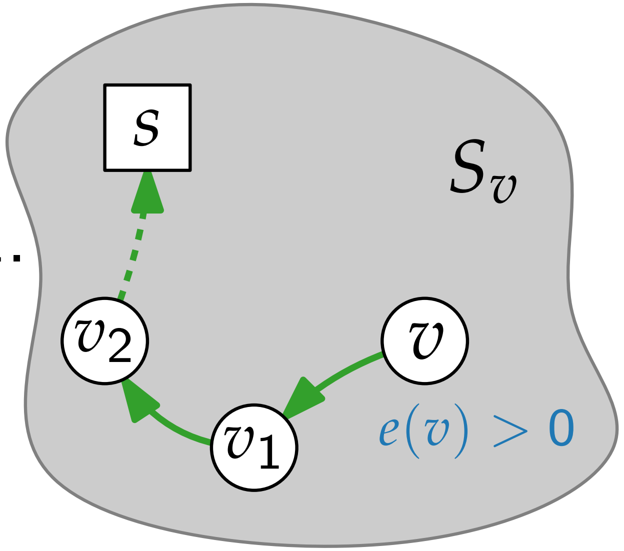
Corollary 8.
The push–relabel algorithm executes
at most $2|V(G)|^2$ Relabel operations.

Height function:

- $h(s) = |V(G)|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1 \quad \forall (u,v) \in E(G_f)$

Condition: u is overflowing and
$h(u) \leq h(v)\ \forall v \in N_f^+(u)$.

Relabel(u)
$h(u) = 1 + \min\{h(v) \colon (u,v) \in E(G_f)\}$



Saturating and Unsaturating Push Operations

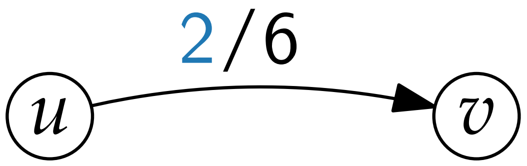
The operation $\textsc{Push}(u, v)$ is

- **saturating** if afterwards $c_f(u, v) = 0$,

Saturating and Unsaturating PUSH Operations

The operation PUSH(u, v) is

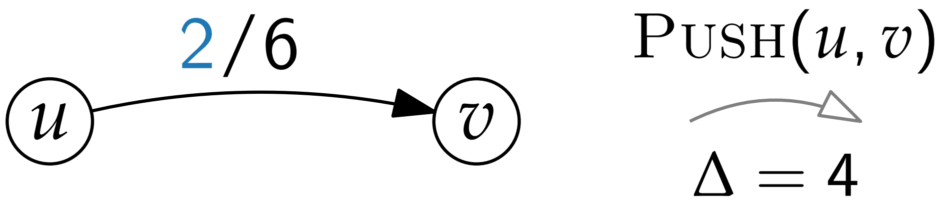
- **saturating** if afterwards $c_f(u, v) = 0$,



Saturating and Unsaturating PUSH Operations

The operation PUSH(u, v) is

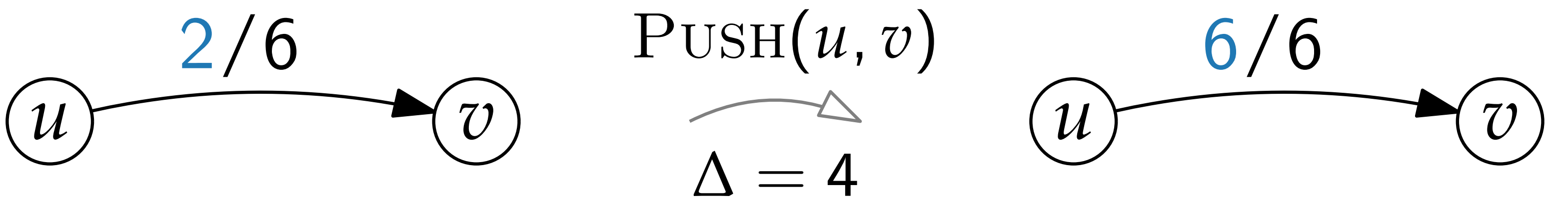
- **saturating** if afterwards $c_f(u, v) = 0$,



Saturating and Unsaturating PUSH Operations

The operation PUSH(u, v) is

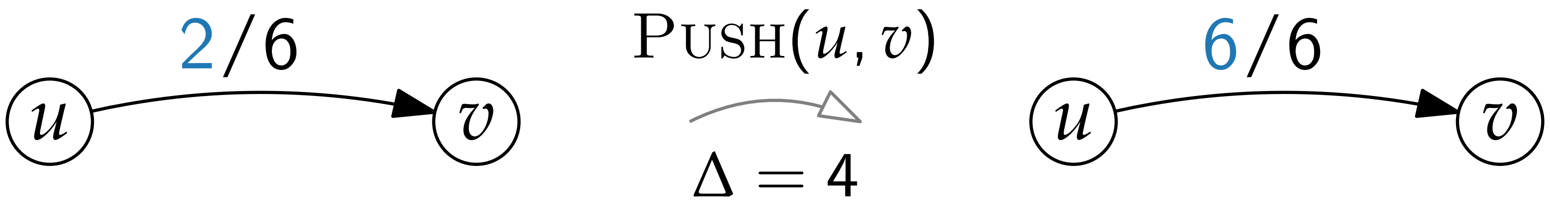
- **saturating** if afterwards $c_f(u, v) = 0$,



Saturating and Unsaturating PUSH Operations

The operation $\textsc{Push}(u, v)$ is

- **saturating** if afterwards $c_f(u, v) = 0$,

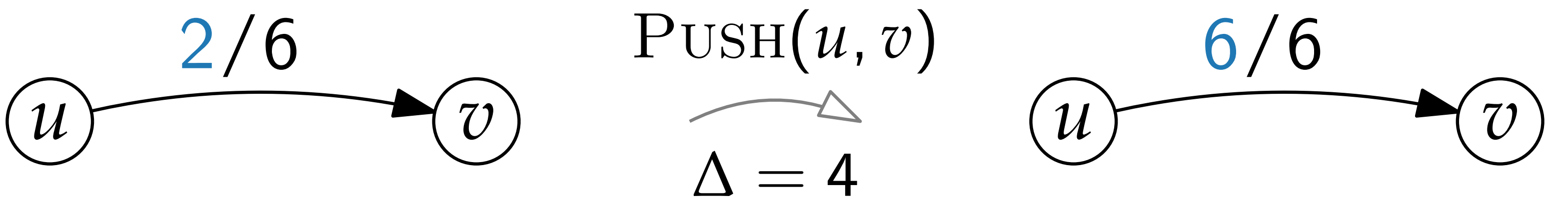


- and **unsaturating** otherwise.

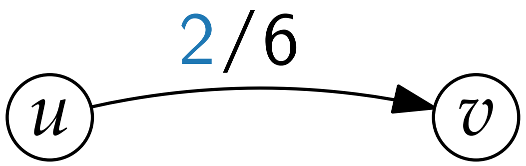
Saturating and Unsaturating $\textsc{Push}$ Operations

The operation $\textsc{Push}(u, v)$ is

- **saturating** if afterwards $c_f(u, v) = 0$,



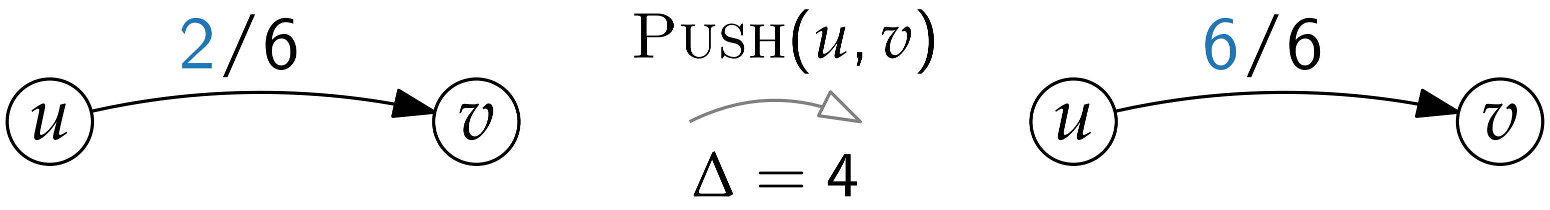
- and **unsaturating** otherwise.



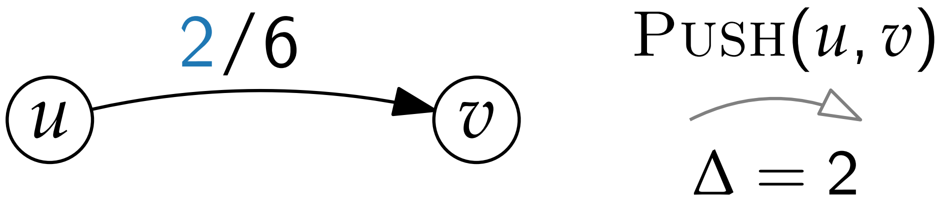
Saturating and Unsaturating PUSH Operations

The operation PUSH(u, v) is

- **saturating** if afterwards $c_f(u, v) = 0$,



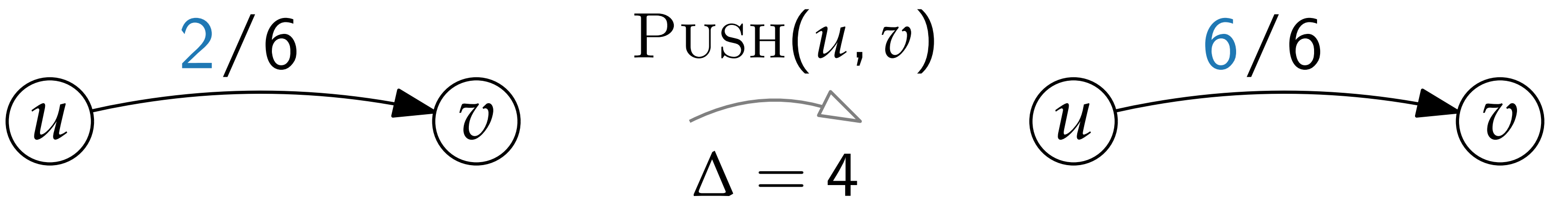
- and **unsaturating** otherwise.



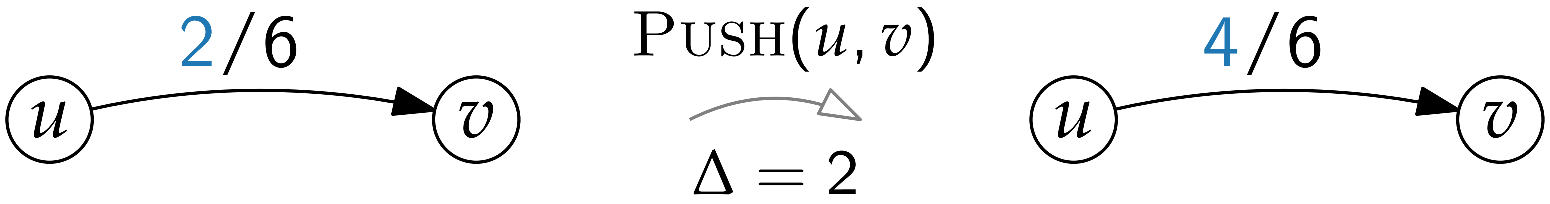
Saturating and Unsaturating PUSH Operations

The operation PUSH(u, v) is

- **saturating** if afterwards $c_f(u, v) = 0$,



- and **unsaturating** otherwise.



Upper Bound on the Number of Saturating PUSH Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating PUSH operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
    f(u, v) = f(u, v) + Δ
  else
    f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Upper Bound on the Number of Saturating PUSH Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating PUSH operations.

Proof.

- Consider saturating PUSH(u, v)
  - $h(u) = h(v) + 1$

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
PUSH(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

PUSH(u, v)
...

Upper Bound on the Number of Saturating Push Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating Push operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Proof.

- Consider saturating Push(u, v)
  - $h(u) = h(v) + 1$
- For another saturating Push(u, v), first Push(v, u) necessary
  - $h(v) = h(u) + 1$ necessary

Push(u, v)
...
Push(v, u)

Upper Bound on the Number of Saturating Push Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating Push operations.

Proof.

- Consider saturating $\textsc{Push}(u, v)$
  - $h(u) = h(v) + 1$
- For another saturating $\textsc{Push}(u, v)$, first $\textsc{Push}(v, u)$ necessary
  - $h(v) = h(u) + 1$ necessary
- After another saturating $\textsc{Push}(u, v)$, both $h(u)$ and $h(v)$ have increased by at least two.

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
      f(u, v) = f(u, v) + Δ
  else
      f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

$\textsc{Push}(u, v)$
...
$\textsc{Push}(v, u)$
...
$\textsc{Push}(u, v)$
...

Upper Bound on the Number of Saturating PUSH Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating PUSH operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
PUSH(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  └   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Proof.

- Consider saturating PUSH(u, v)
  - $h(u) = h(v) + 1$
- For another saturating PUSH(u, v), first PUSH(v, u) necessary
  - $h(v) = h(u) + 1$ necessary
- After another saturating PUSH(u, v), both $h(u)$ and $h(v)$ have increased by at least two.
- But by Lemma 7, $h(u) \leq 2|V(G)| - 1$ and $h(v) \leq 2|V(G)| - 1$.

PUSH(u, v)
...
PUSH(v, u)
...
PUSH(u, v)
...

Upper Bound on the Number of Saturating PUSH Operations

Lemma 9.
The push–relabel algorithm executes at most $2|V(G)| \cdot |E(G)|$ saturating PUSH operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
PUSH(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |_  f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Proof.

- Consider saturating PUSH(u, v)
  - $h(u) = h(v) + 1$
- For another saturating PUSH(u, v), first PUSH(v, u) necessary
  - $h(v) = h(u) + 1$ necessary
- After another saturating PUSH(u, v), both $h(u)$ and $h(v)$ have increased by at least two.
- But by Lemma 7, $h(u) \leq 2|V(G)| - 1$ and $h(v) \leq 2|V(G)| - 1$.
- There are at most $2|V(G)| - 1$ saturating PUSH operations for every edge (u, v).

PUSH(u, v)
...
PUSH(v, u)
...
PUSH(u, v)
...

Upper Bound on the Number of Unsaturating PUSH Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating PUSH operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u, v)

$\Delta = \min(e(u), c_f(u,v))$
if $(u,v) \in E(G)$ **then**
  $f(u,v) = f(u,v) + \Delta$
else
  $f(v,u) = f(v,u) - \Delta$

$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Upper Bound on the Number of Unsaturating PUSH Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating PUSH operations.

Proof.

- Consider $\mathcal{H} = \sum_{\substack{v \in V(G) \setminus \{s,t\}, \\ v \text{ overflowing}}} h(v)$.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u,v)

$\Delta = \min(e(u), c_f(u,v))$
if $(u,v) \in E(G)$ **then**
  $f(u,v) = f(u,v) + \Delta$
else
  $f(v,u) = f(v,u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Upper Bound on the Number of Unsaturating Push Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating Push operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u,v))
  if (u,v) ∈ E(G) then
  |  f(u,v) = f(u,v) + Δ
  else
  |  f(v,u) = f(v,u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Proof.

- Consider $\mathcal{H} = \sum_{\substack{v \in V(G) \setminus \{s,t\}, \\ v \text{ overflowing}}} h(v)$.
- After initialization and at the end $\mathcal{H} = 0$.
- A saturating Push increases $\mathcal{H}$ by at most $2|V| - 1$.
- By Lemma 9, all saturating Push operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot 2|V||E|$.

Upper Bound on the Number of Unsaturating Push Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating Push operations.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
Push(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
  |   f(u, v) = f(u, v) + Δ
  else
  |   f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Proof.

- Consider $\mathcal{H} = \sum_{\substack{v \in V(G) \setminus \{s,t\}, \\ v \text{ overflowing}}} h(v)$.
- After initialization and at the end $\mathcal{H} = 0$.
- A saturating Push increases $\mathcal{H}$ by at most $2|V| - 1$.
- By Lemma 9, all saturating Push operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot 2|V||E|$.
- By Lemma 7, all Relabel operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot |V|$.

Upper Bound on the Number of Unsaturating PUSH Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating PUSH operations.

Proof.

- Consider $\mathcal{H} = \sum_{\substack{v \in V(G) \setminus \{s,t\}, \\ v \text{ overflowing}}} h(v)$.
- After initialization and at the end $\mathcal{H} = 0$.
- A saturating PUSH increases $\mathcal{H}$ by at most $2|V| - 1$.
- By Lemma 9, all saturating PUSH operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot 2|V||E|$.
- By Lemma 7, all RELABEL operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot |V|$.
- An unsaturating PUSH(u, v) decreases $\mathcal{H}$ by at least 1 since $h(u) - h(v) \geq 1$.

Condition: u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

```
PUSH(u, v)
  Δ = min(e(u), c_f(u, v))
  if (u, v) ∈ E(G) then
    | f(u, v) = f(u, v) + Δ
  else
    └ f(v, u) = f(v, u) − Δ
  e(u) = e(u) − Δ
  e(v) = e(v) + Δ
```

Upper Bound on the Number of Unsaturating PUSH Ops

Lemma 10.
The push–relabel algorithm executes at most $4|V(G)|^2 \cdot |E(G)|$ unsaturating PUSH operations.

Proof.

- Consider $\mathcal{H} = \sum_{\substack{v \in V(G)\setminus\{s,t\}, \\ v \text{ overflowing}}} h(v)$.
- After initialization and at the end $\mathcal{H} = 0$.
- A saturating PUSH increases $\mathcal{H}$ by at most $2|V| - 1$.
- By Lemma 9, all saturating PUSH operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot 2|V||E|$.
- By Lemma 7, all RELABEL operations increase $\mathcal{H}$ by $\leq (2|V| - 1) \cdot |V|$.
- An unsaturating PUSH(u, v) decreases $\mathcal{H}$ by at least 1 since $h(u) - h(v) \geq 1$.

$\Rightarrow$ At most $4|V|^2|E| + 2|V|^2 - 2|V||E| - |V|$ unsaturating PUSH operations.

Condition: u is overflowing, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

PUSH(u, v)
$\Delta = \min(e(u), c_f(u, v))$
if $(u, v) \in E(G)$ **then**
  $f(u, v) = f(u, v) + \Delta$
else
  $f(v, u) = f(v, u) - \Delta$
$e(u) = e(u) - \Delta$
$e(v) = e(v) + \Delta$

Termination of the Algorithm

Theorem 5.
When the push–relabel algorithm terminates, the computed preflow f is a maximum flow.

Theorem 11.
The push–relabel algorithm terminates after $\mathcal{O}(|V|^2|E|)$ valid $\textsc{Push}$ or $\textsc{Relabel}$ ops.

Proof.

- Follows by Corollary 8 and Lemmas 9 & 10.

Implementation

The actual running time depends on the selection order of the overflowing vertices:

- **FIFO implementation:**
  Pick overflowing vertex by *first-in-first-out* principle: $\mathcal{O}(|V|^3)$ running time.
  with dynamic trees: $\mathcal{O}(|V||E| \log \frac{|V|^2}{|E|})$ time.

- **Highest label:**
  For $\textsc{Push}$ select highest overflowing vertex: $\mathcal{O}(|V|^2|E|^{\frac{1}{2}})$ time.

- **Excess scaling:**
  For $\textsc{Push}(u, v)$ choose edge (u, v) such that u is overflowing, $e(u)$ is *sufficiently high* and $e(v)$ *sufficiently small*: $\mathcal{O}(|E| + |V|^2 \log C)$ time, where $C = \max_{e \in E} c(e)$.

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\lvert E\rvert \lvert V\rvert \max_{e\in E} c(e))$

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \max_{e \in E} c(e))$
Edmonds, Karp	1969	augmenting paths	$\mathcal{O}(\lvert E\rvert^2\lvert V\rvert)$

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \max_{e\in E} c(e))$
Edmonds, Karp	1969	augmenting paths	$\mathcal{O}(\lvert E\rvert^2\lvert V\rvert)$
Dinic [version w/ dynamic trees]	1970	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \log \lvert V\rvert)$

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \max_{e\in E} c(e))$
Edmonds, Karp	1969	augmenting paths	$\mathcal{O}(\lvert E\rvert^2\lvert V\rvert)$
Dinic [version w/ dynamic trees]	1970	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \log\lvert V\rvert)$
Goldberg, Tarjan [v. w/ dynamic trees]	1988	push–relabel	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \log(\lvert V\rvert^2/\lvert E\rvert))$

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\|E\|\|V\| \max_{e \in E} c(e))$
Edmonds, Karp	1969	augmenting paths	$\mathcal{O}(\|E\|^2\|V\|)$
Dinic [version w/ dynamic trees]	1970	augmenting paths	$\mathcal{O}(\|E\|\|V\| \log \|V\|)$
Goldberg, Tarjan [v. w/ dynamic trees]	1988	push–relabel	$\mathcal{O}(\|E\|\|V\| \log(\|V\|^2/\|E\|))$
Orlin, King, Rao, Tarjan	2013	push–relabel & more	$\mathcal{O}(\|E\|\|V\|)$

Discussion

- The push–relabel method offers an alternative and faster framework to the augmenting-paths method to develop algorithms that solve the maximum-flow problem.
- In practice, heuristics are used to improve the performance of push–relabel algorithms. Any ideas?
- The algorithm can be extended to solve the minimum-cost flow problem.
- Meanwhile there are even faster (and more complicated) algorithms for the maximum-flow problem known:

Authors	Year	Basic technique	Running time
Ford, Fulkerson	1956	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \max_{e\in E} c(e))$
Edmonds, Karp	1969	augmenting paths	$\mathcal{O}(\lvert E\rvert^2\lvert V\rvert)$
Dinic [version w/ dynamic trees]	1970	augmenting paths	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \log \lvert V\rvert)$
Goldberg, Tarjan [v. w/ dynamic trees]	1988	push–relabel	$\mathcal{O}(\lvert E\rvert\lvert V\rvert \log(\lvert V\rvert^2/\lvert E\rvert))$
Orlin, King, Rao, Tarjan	2013	push–relabel & more	$\mathcal{O}(\lvert E\rvert\lvert V\rvert)$
Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva	2022	interior point method	$\mathcal{O}(\lvert E\rvert^{1+o(1)} \log \max_{e\in E} c(e))$

Literature

Main source:

- [CLRS Ch26] $\leftarrow$ Cormen et al. “Introduction to Algorithms”

Original paper:

- [Goldberg, Tarjan ’88] A new approach to the maximum-flow problem

Links:

- Animations of the max-flow algorithms by Ford–Fulkerson and Edmonds–Karp: https://visualgo.net/en/maxflow