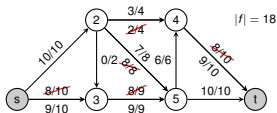
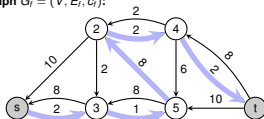


Graph $G = (V, E, c)$:



Residual Graph $G_r = (V, E_r, c_r)$:



6.6: Maximum flow

Frank Stajano

Thomas Sauerwald

Lent 2016



UNIVERSITY OF
CAMBRIDGE

Outline

Introduction

Ford-Fulkerson

A Glimpse at the Max-Flow Min-Cut Theorem

Analysis of Ford-Fulkerson



History of the Maximum Flow Problem [Harris, Ross (1955)]

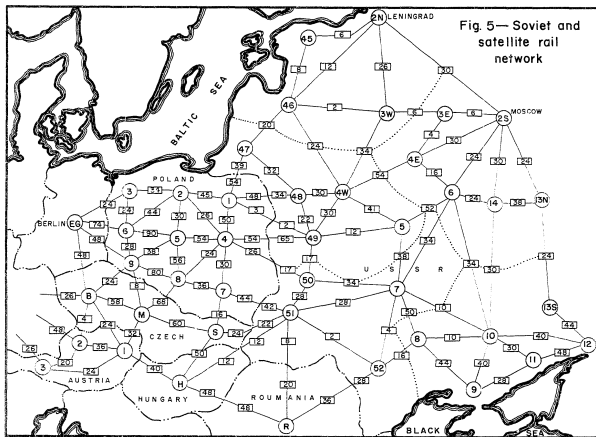


Fig. 5— Soviet and satellite rail network

Legend: — International boundary Regional boundaries of the USSR (they are included as a matter of general information)

⑦ Operating divisions. Those located in Russia are believed to be accurately located. Some Russian divisions (2, 3, 4 and 13) are located in two regions and are so indicated. Divisions shown in the satellites are indicated according to the authors' best judgment, since intelligence reports are unavailable. Train capacities in Russia are for 1000-net-ton trains or their equivalent. Train capacities in Poland are for 665 net tons (or the equivalent). Train capacities in all other satellites are for 400 net tons (or the equivalent) except in East Germany, train capacities are those of entering lines. The numbers shown in boxes are total interdivisional capacities.



History of the Maximum Flow Problem [Harris, Ross (1955)]

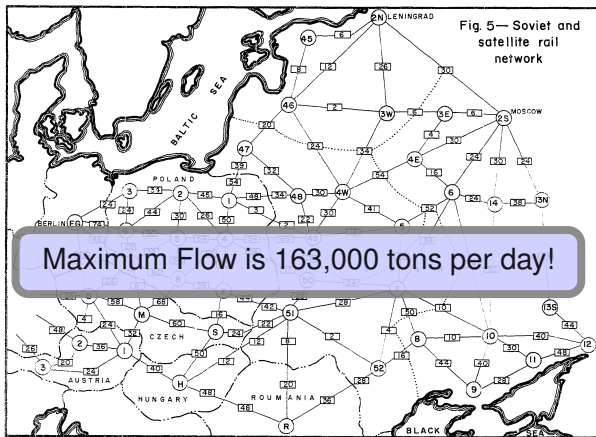


Fig. 5— Soviet and satellite rail network

Maximum Flow is 163,000 tons per day!

Legend: — International boundary Regional boundaries of the USSR (they are included as a matter of general information)

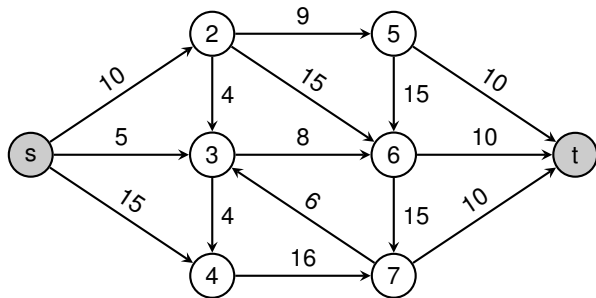
⑦ Operating divisions. Those located in Russia are believed to be accurately located. Some Russian divisions (2, 3, 4 and 13) are located in two regions and are so indicated. Divisions shown in the satellites are indicated according to the authors' best judgment, since intelligence reports are unavailable. Train capacities in Russia are for 1000-net-ton trains or their equivalent. Train capacities in Poland are for 665 net tons (or the equivalent). Train capacities in all other satellites are for 400 net tons (or the equivalent) except in East Germany, in East Germany, train capacities are those of entering lines. The numbers shown in boxes are total interdivisional capacities.



Flow Network

Flow Network

- Abstraction for material (one commodity!) **flowing** through the edges
- $G = (V, E)$ directed graph **without parallel edges**
- distinguished nodes: source s and sink t
- every edge e has a capacity $c(e)$

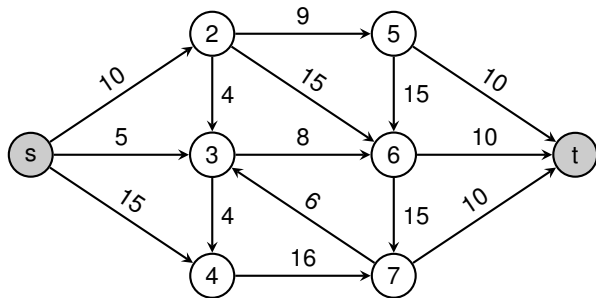


Flow Network

Flow Network

- Abstraction for material (one commodity!) **flowing** through the edges
- $G = (V, E)$ directed graph **without parallel edges**
- distinguished nodes: source s and sink t
- every edge e has a capacity $c(e)$

Capacity function $c : V \times V \rightarrow \mathbb{R}^+$



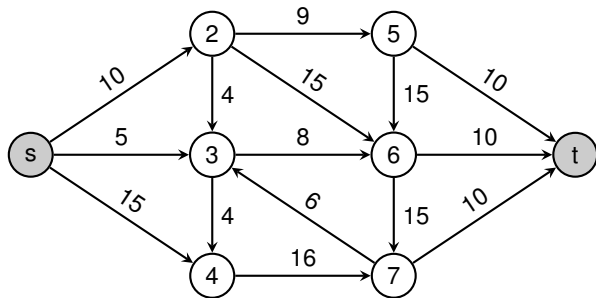
Flow Network

Flow Network

- Abstraction for material (one commodity!) **flowing** through the edges
- $G = (V, E)$ directed graph **without parallel edges**
- distinguished nodes: source s and sink t
- every edge e has a capacity $c(e)$

Capacity function $c : V \times V \rightarrow \mathbb{R}^+$

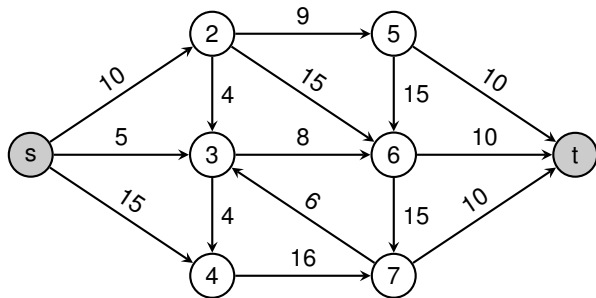
$c(u, v) = 0 \Leftrightarrow (u, v) \notin E$



Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

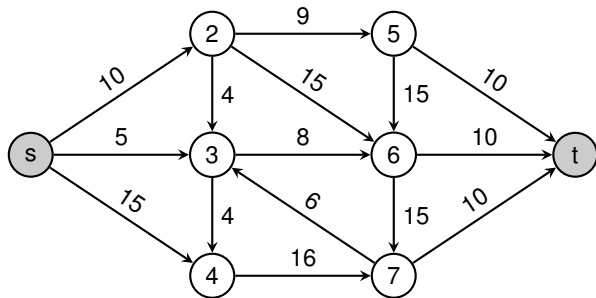


Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$

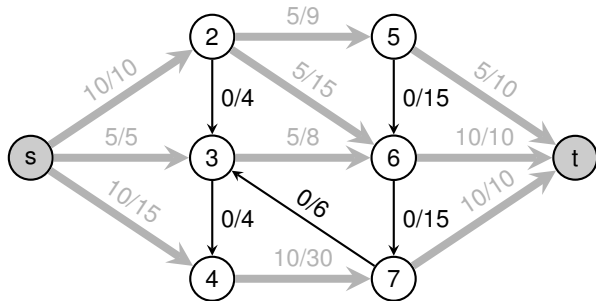


Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$

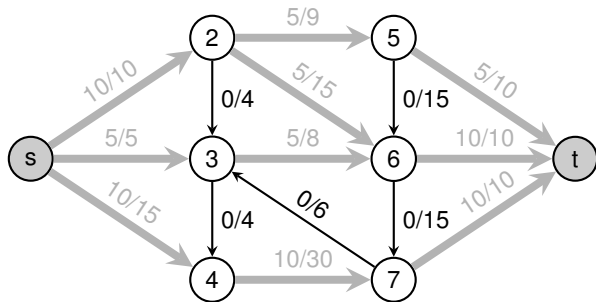


Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$

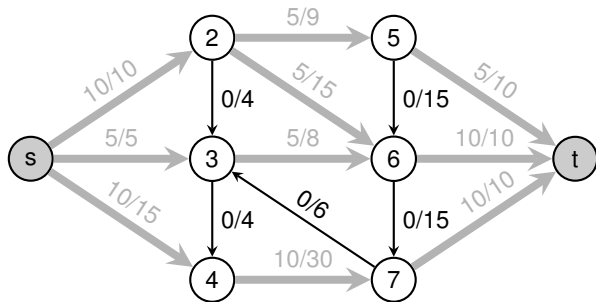


Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$



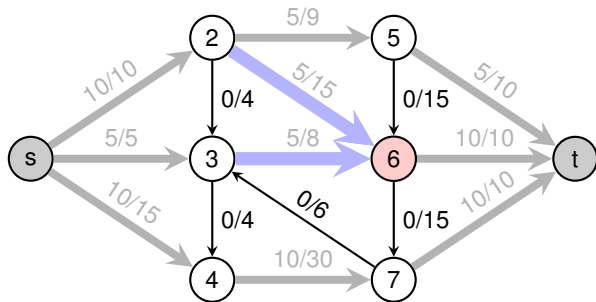
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

Flow Conservation



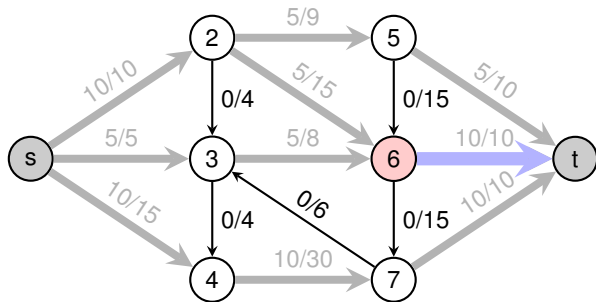
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

Flow Conservation



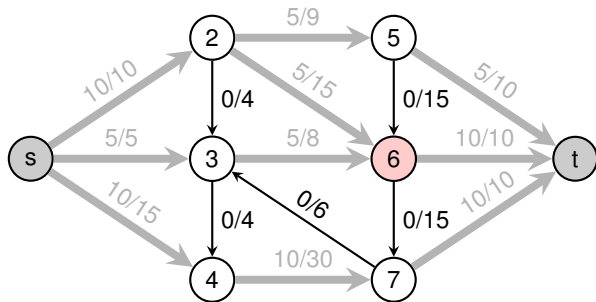
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

Flow Conservation



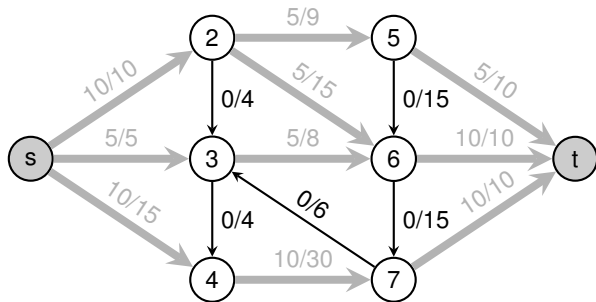
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

Flow Conservation



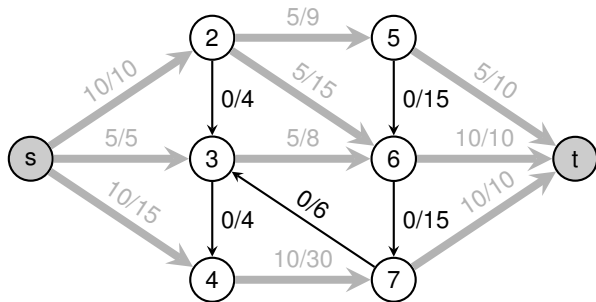
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$



Flow Network

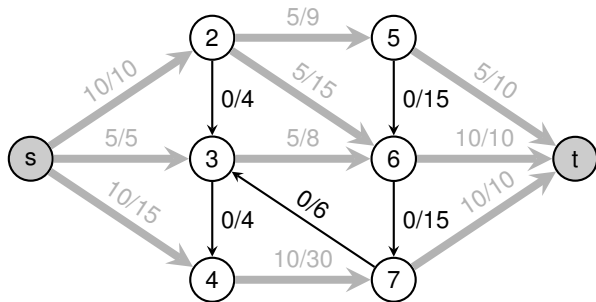
Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$

$$\sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$



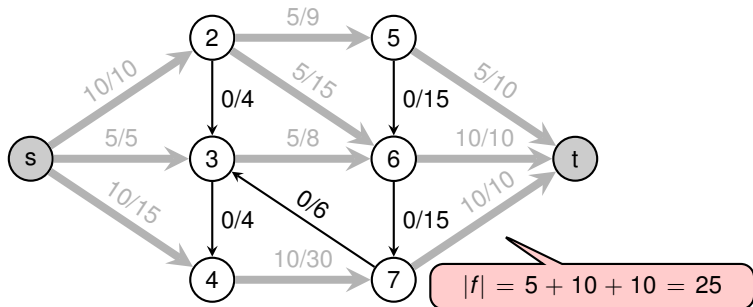
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$



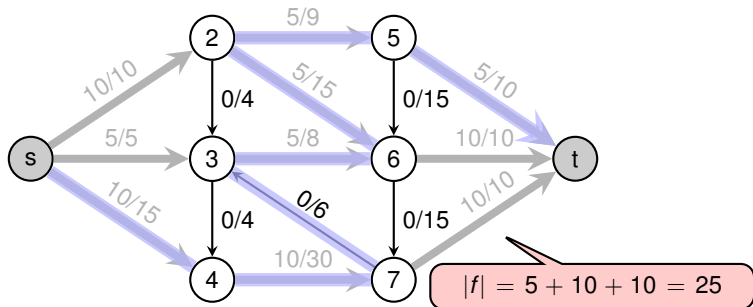
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$



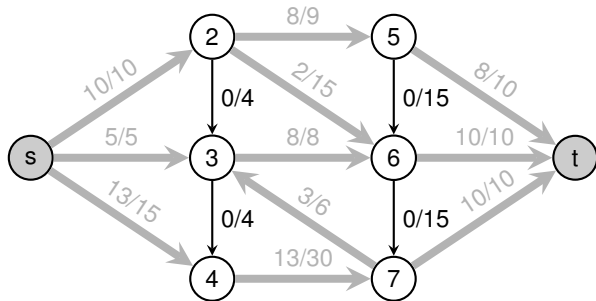
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$



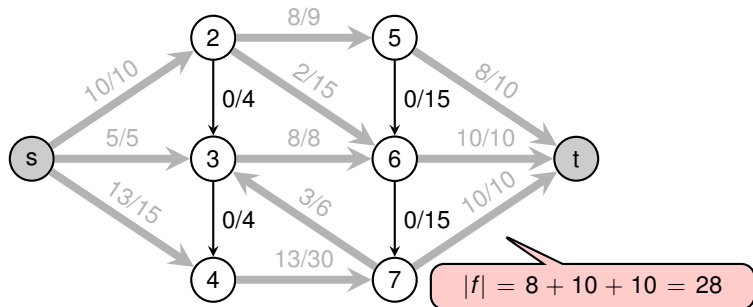
Flow Network

Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$



Flow Network

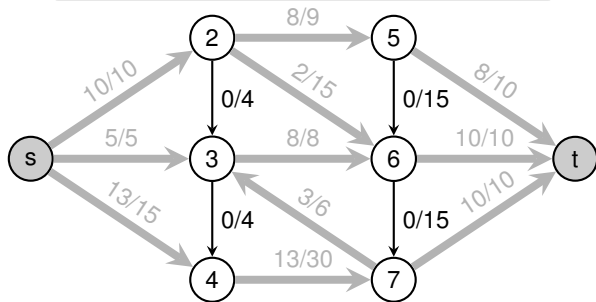
Flow

A **flow** is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

- For every $u, v \in V$, $f(u, v) \leq c(u, v)$
- For every $u, v \in V$, $f(u, v) = -f(v, u)$
- For every $u \in V \setminus \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

The **value** of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$

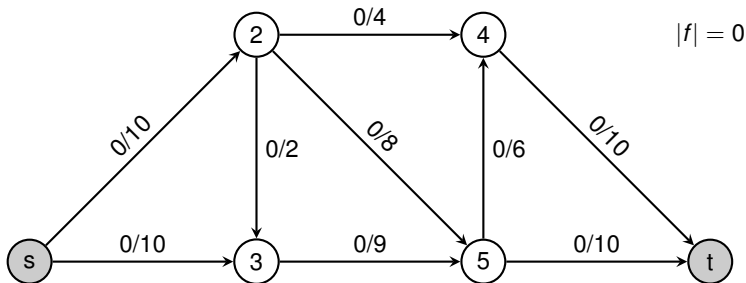
How to find a Maximum Flow?



A First Attempt

Greedy Algorithm

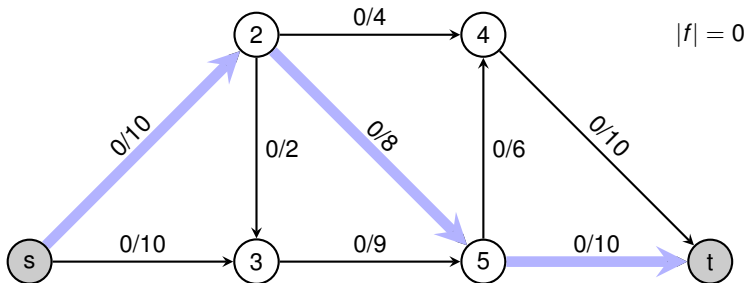
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

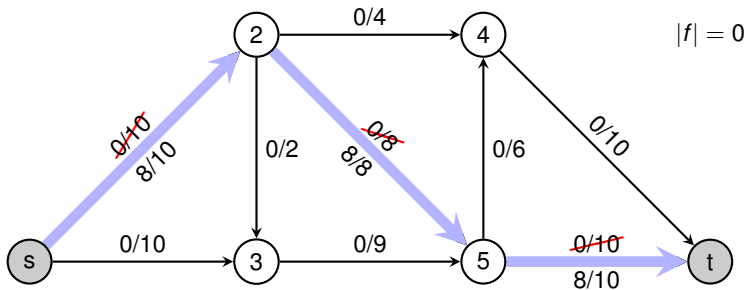
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

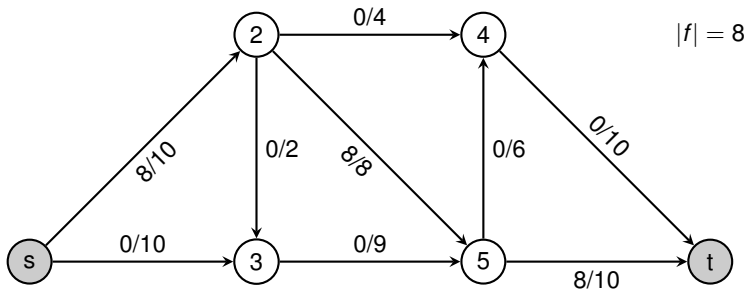
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

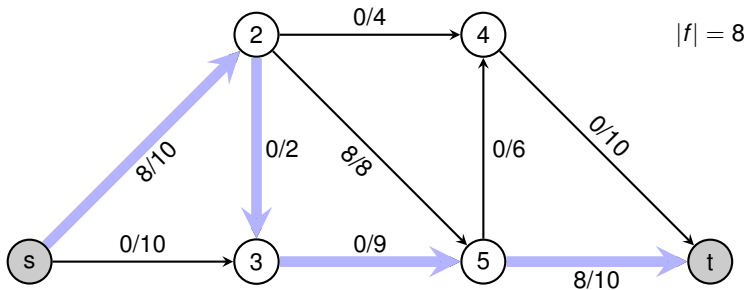
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

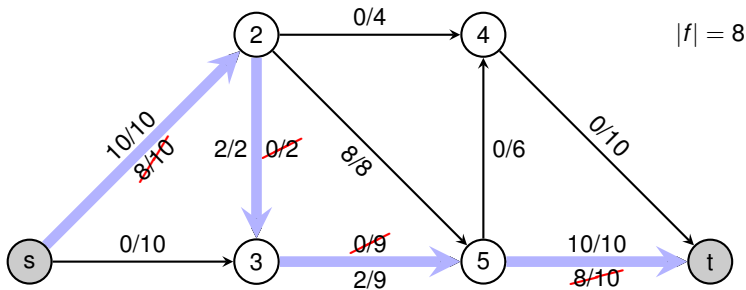
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

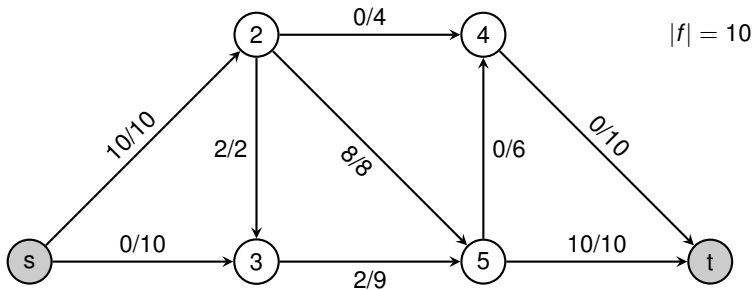
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

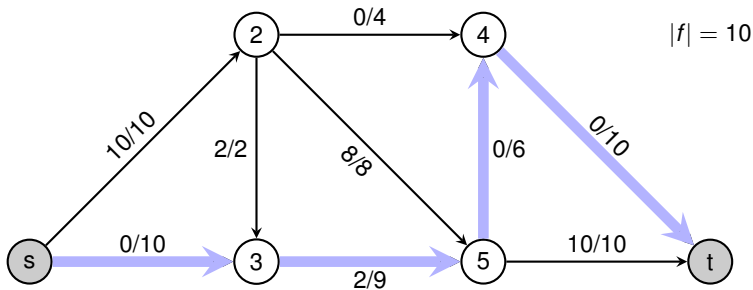
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

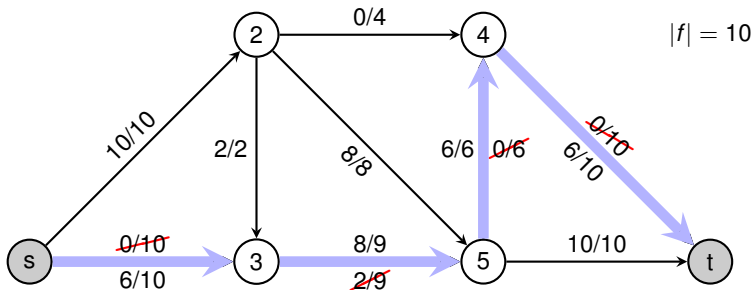
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

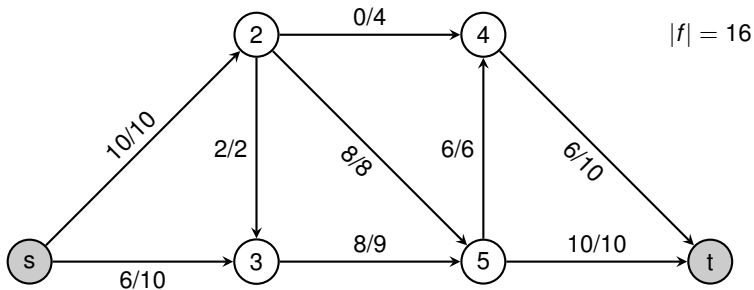
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

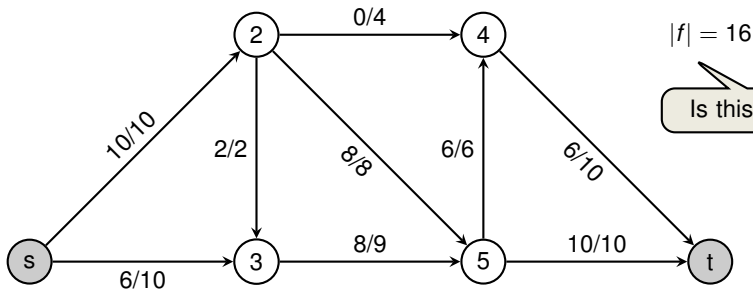
- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



A First Attempt

Greedy Algorithm

- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



$$|f| = 16$$

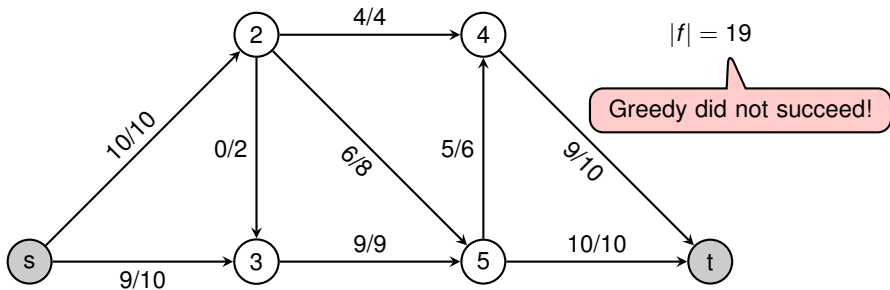
Is this optimal?



A First Attempt

Greedy Algorithm

- Start with $f(u, v) = 0$ everywhere
- Repeat as long as possible:
 - Find a (s, t) -path p where each edge $e = (u, v)$ has $f(u, v) < c(u, v)$
 - Augment flow along p



Introduction

Ford-Fulkerson

A Glimpse at the Max-Flow Min-Cut Theorem

Analysis of Ford-Fulkerson



Residual Graph

Original Edge

Edge $e = (u, v) \in E$

- flow $f(u, v)$ and capacity $c(u, v)$



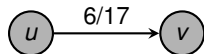
Residual Graph

Original Edge

Edge $e = (u, v) \in E$

- flow $f(u, v)$ and capacity $c(u, v)$

Graph G:



Residual Graph

Original Edge

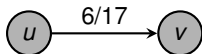
Edge $e = (u, v) \in E$

- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

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

Graph G:



Residual Graph

Original Edge

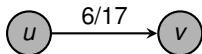
Edge $e = (u, v) \in E$

- flow $f(u, v)$ and capacity $c(u, v)$

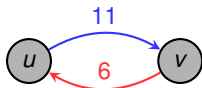
Residual Capacity

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

Graph G :



Residual G_f :



Residual Graph

Original Edge

Edge $e = (u, v) \in E$

- flow $f(u, v)$ and capacity $c(u, v)$

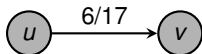
Residual Capacity

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

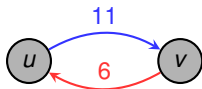
Residual Graph

- $G_f = (V, E_f, c_f)$, $E_f := \{(u, v) : c_f(u, v) > 0\}$

Graph G :



Residual G_f :



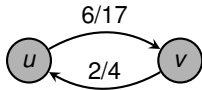
Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

- flow $f(u, v)$ and capacity $c(u, v)$

Graph G:



Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

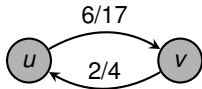
- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

For every pair $(u, v) \in V \times V$,

$$c_f(u, v) = c(u, v) - f(u, v).$$

Graph G:



Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

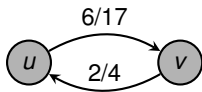
- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

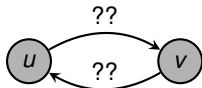
For every pair $(u, v) \in V \times V$,

$$c_f(u, v) = c(u, v) - f(u, v).$$

Graph G:



Residual G_f :



Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

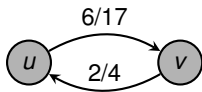
- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

For every pair $(u, v) \in V \times V$,

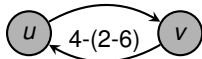
$$c_f(u, v) = c(u, v) - f(u, v).$$

Graph G:



Residual G_f :

$17 - (6 - 2)$



Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

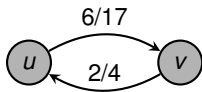
- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

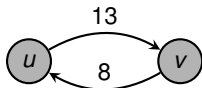
For every pair $(u, v) \in V \times V$,

$$c_f(u, v) = c(u, v) - f(u, v).$$

Graph G :



Residual G_f :



Residual Graph with anti-parallel edges

Original Edge

Edge $e = (u, v) \in E$ (& possibly $e' = (v, u) \in E$)

- flow $f(u, v)$ and capacity $c(u, v)$

Residual Capacity

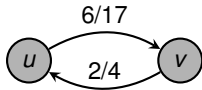
For every pair $(u, v) \in V \times V$,

$$c_f(u, v) = c(u, v) - f(u, v).$$

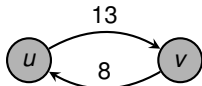
Residual Graph

- $G_f = (V, E_f, c_f)$, $E_f := \{(u, v) : c_f(u, v) > 0\}$

Graph G:

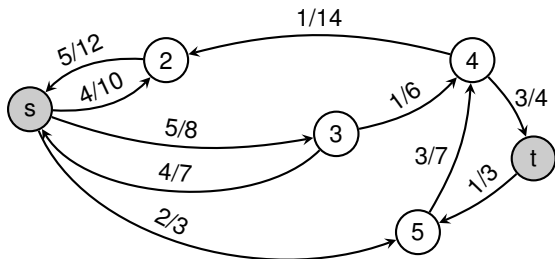


Residual G_f :

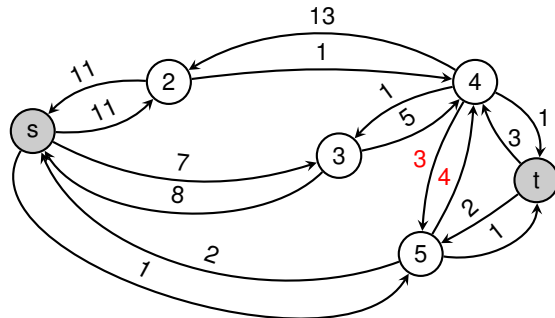


Example of a Residual Graph (Handout)

Flow network G

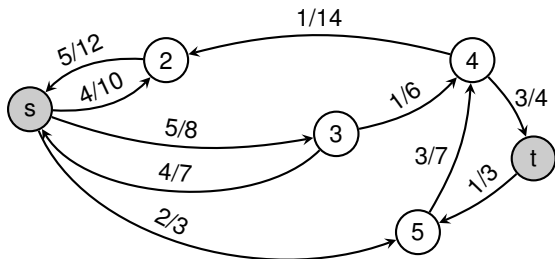


Residual Graph G_f



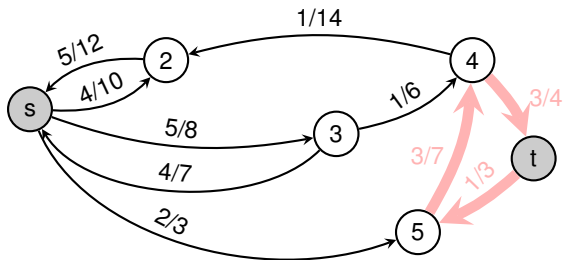
Example of a Residual Graph (Handout)

Flow network G



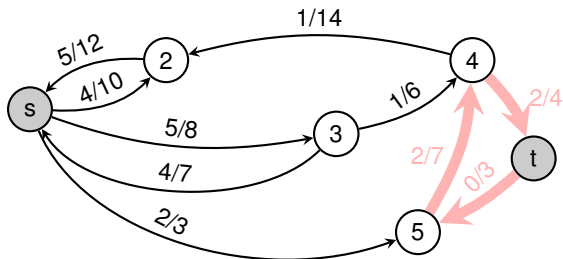
Example of a Residual Graph (Handout)

Flow network G



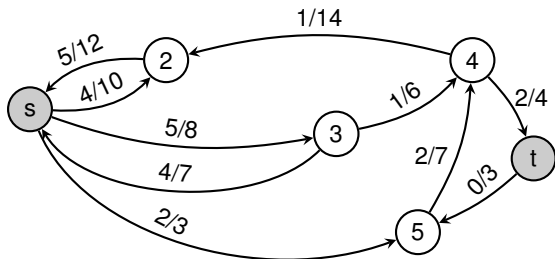
Example of a Residual Graph (Handout)

Flow network G



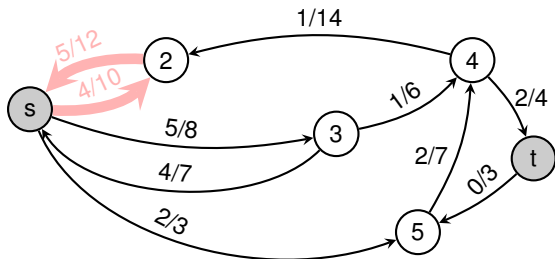
Example of a Residual Graph (Handout)

Flow network G



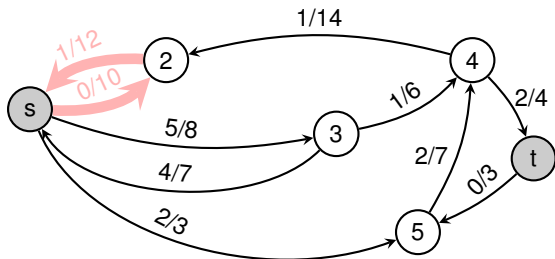
Example of a Residual Graph (Handout)

Flow network G



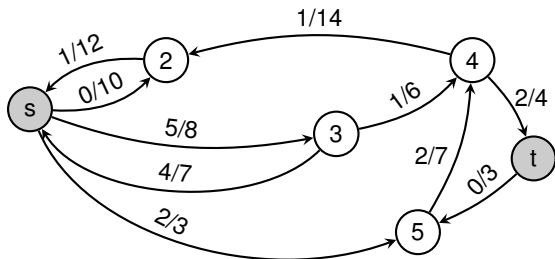
Example of a Residual Graph (Handout)

Flow network G



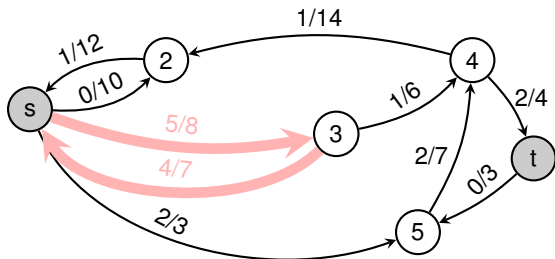
Example of a Residual Graph (Handout)

Flow network G



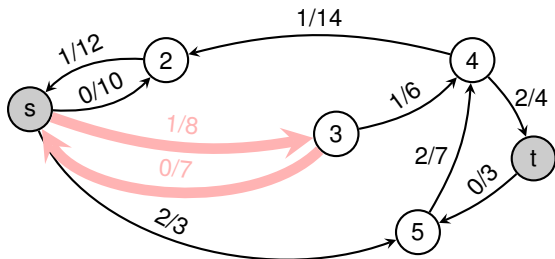
Example of a Residual Graph (Handout)

Flow network G



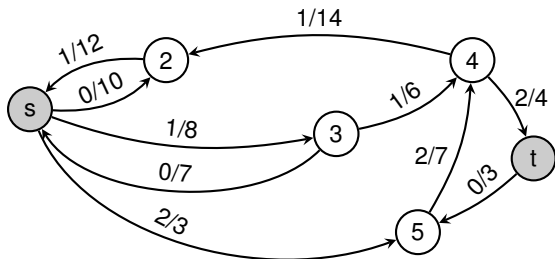
Example of a Residual Graph (Handout)

Flow network G



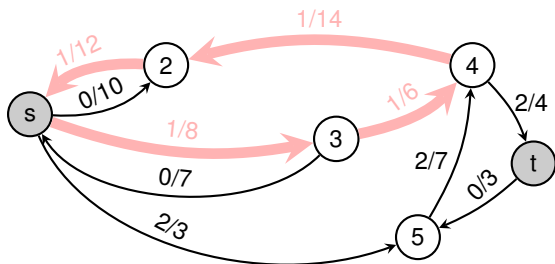
Example of a Residual Graph (Handout)

Flow network G



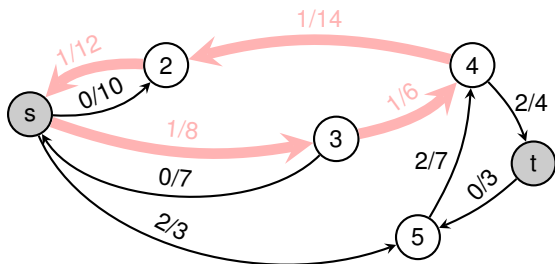
Example of a Residual Graph (Handout)

Flow network G



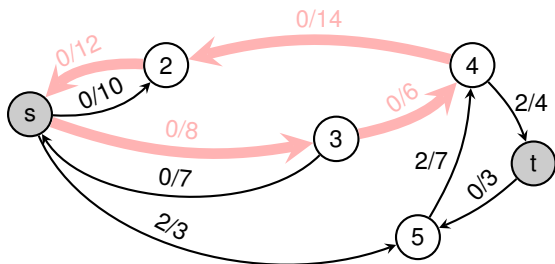
Example of a Residual Graph (Handout)

Flow network G



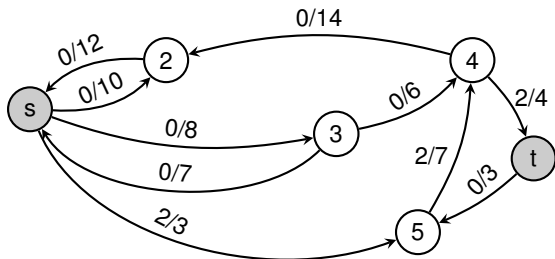
Example of a Residual Graph (Handout)

Flow network G



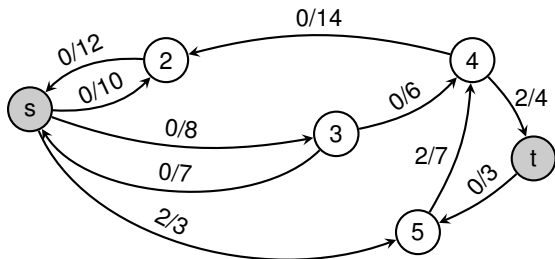
Example of a Residual Graph (Handout)

Flow network G



Example of a Residual Graph (Handout)

Flow network G



By successively eliminating cycles we can simplify and reduce the “transportation” cost of a flow.



The Ford-Fulkerson Method (“Enhanced Greedy”)

```
0: def fordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```



The Ford-Fulkerson Method (“Enhanced Greedy”)

```
0: def fordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Augmenting path: Path
from source to sink in G_f



The Ford-Fulkerson Method (“Enhanced Greedy”)

```
0: def fordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

If f' is a flow in G_f and f a flow in G , then $f + f'$ is a flow in G



The Ford-Fulkerson Method (“Enhanced Greedy”)

```
0: def fordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Questions:

- How to find an augmenting path?
- Does this method terminate?
- If it terminates, how good is the solution?



The Ford-Fulkerson Method (“Enhanced Greedy”)

```
0: def fordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Using BFS or DFS, we can find an augmenting path in $O(V + E)$ time.

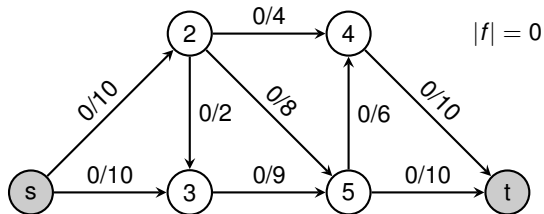
Questions:

- How to find an augmenting path?
- Does this method terminate?
- If it terminates, how good is the solution?



Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

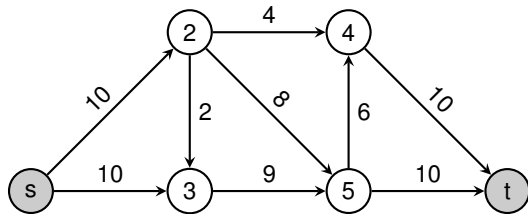
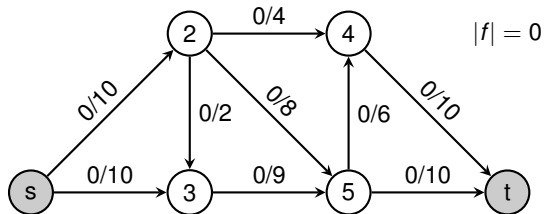


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

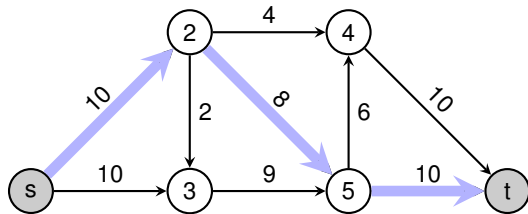
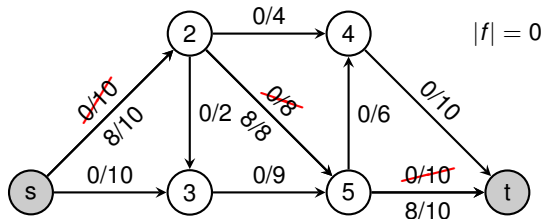


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

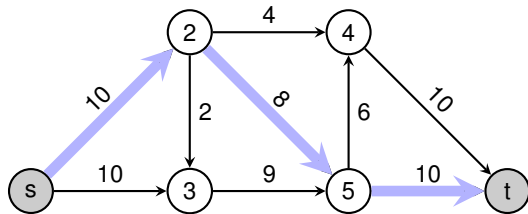
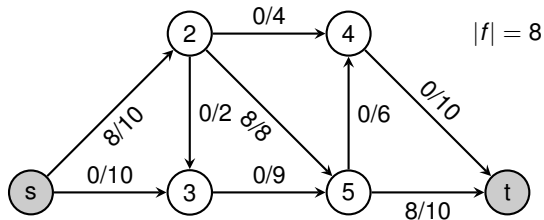


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

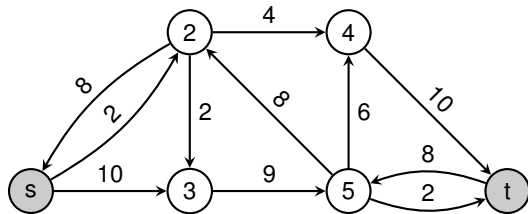
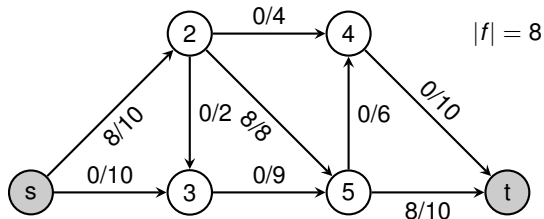


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

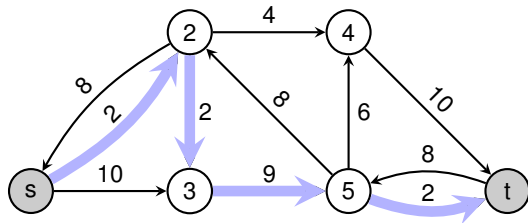
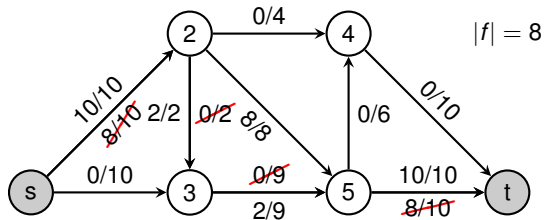


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

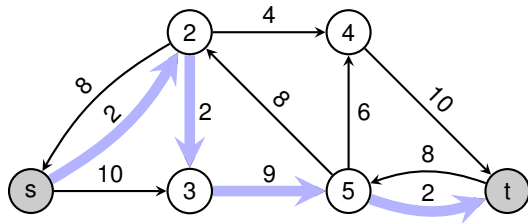
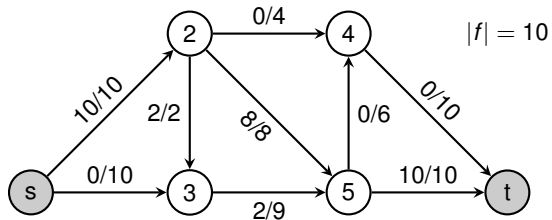


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

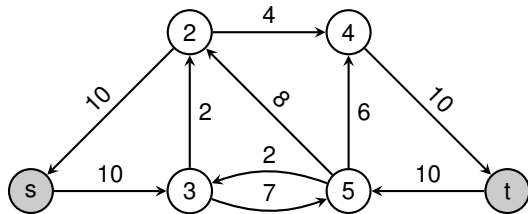
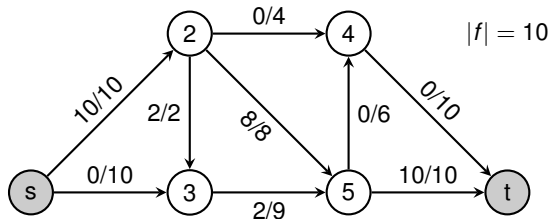


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

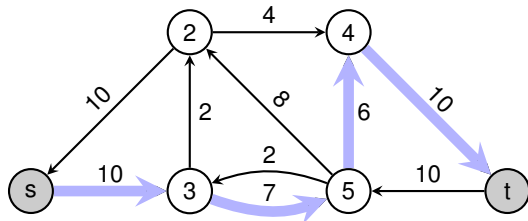
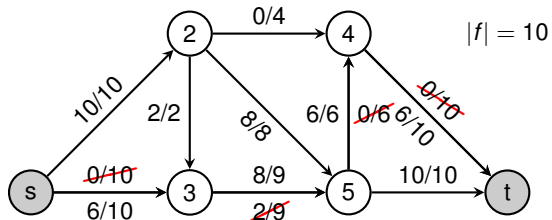


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

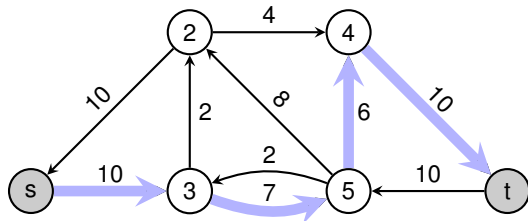
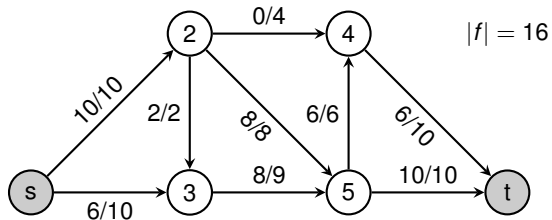


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

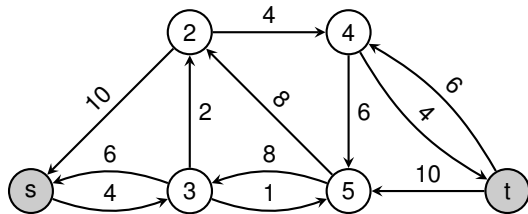
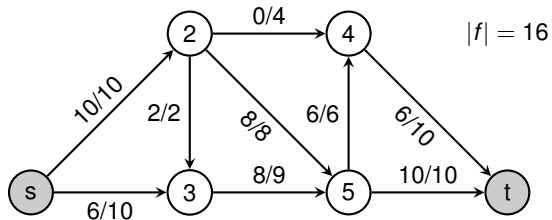


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

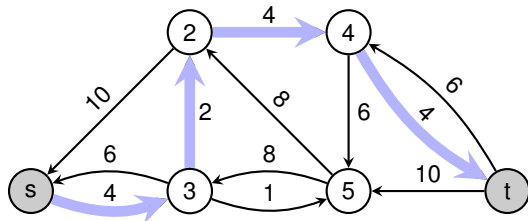
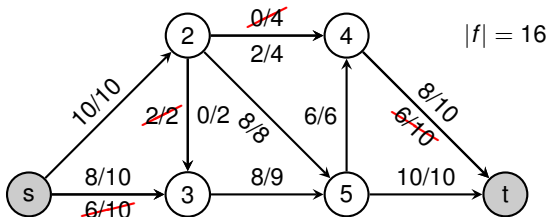


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

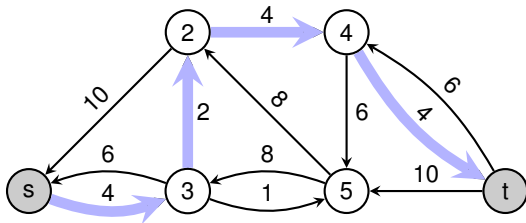
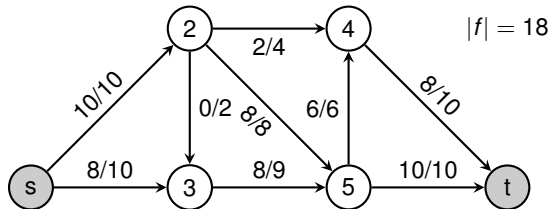


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

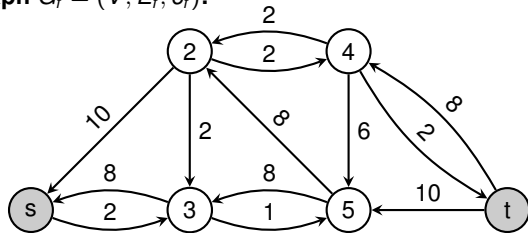
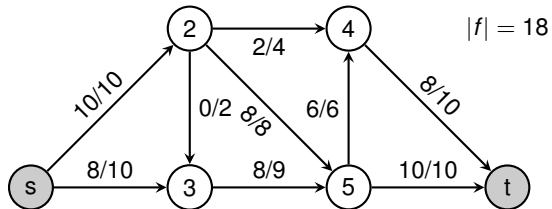


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

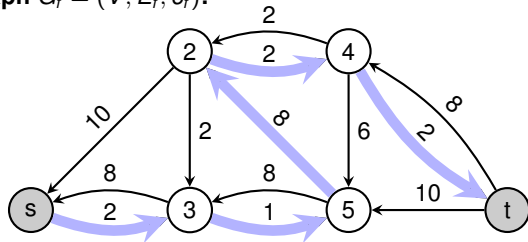
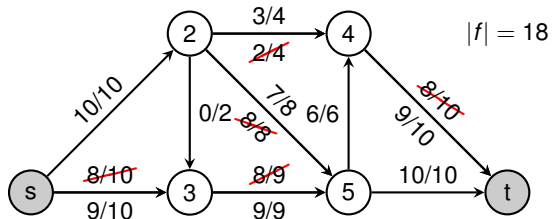


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

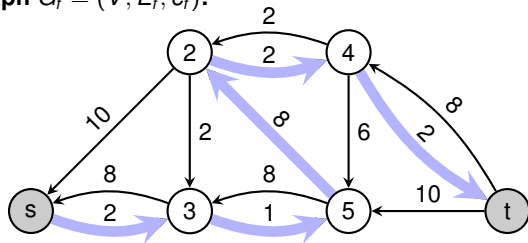
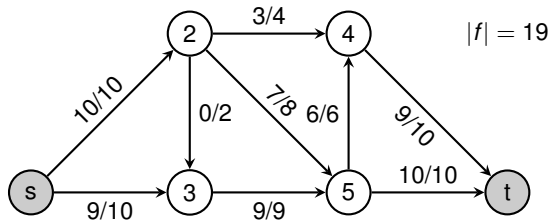


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

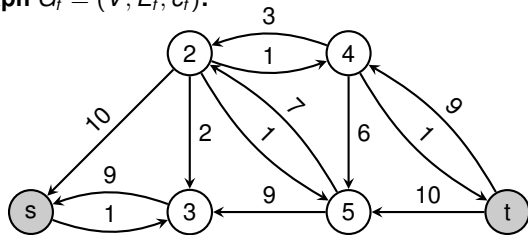
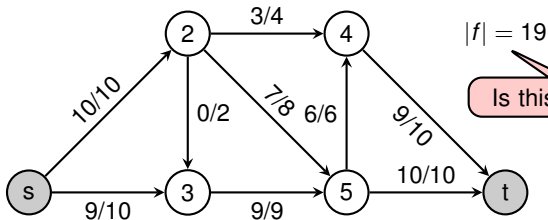


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Is this a max-flow?

Residual Graph $G_f = (V, E_f, c_f)$:

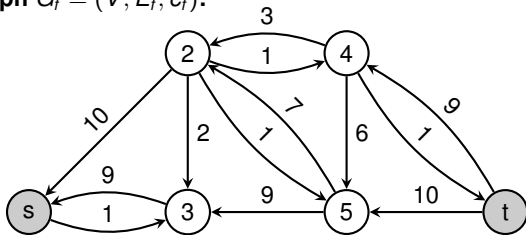
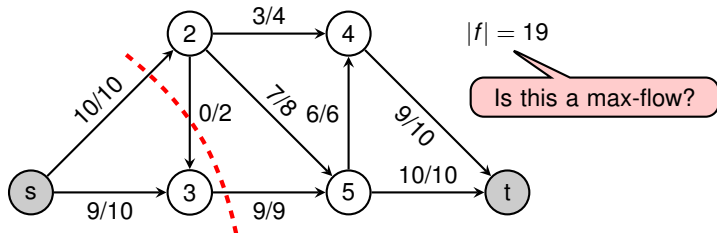


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:

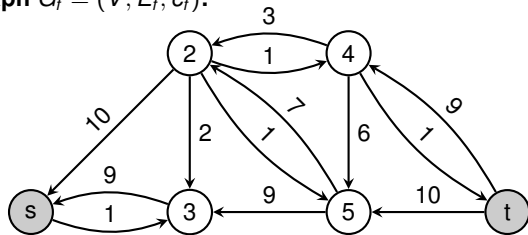
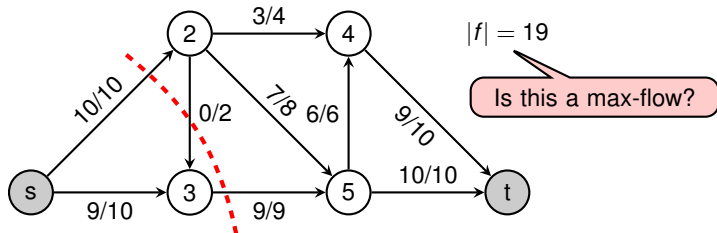
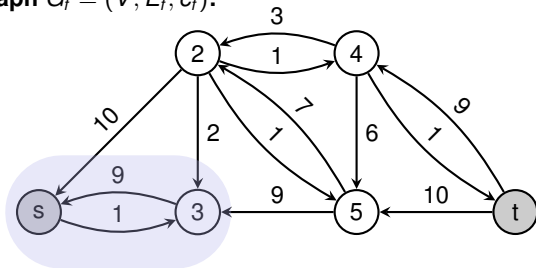


Illustration of the Ford-Fulkerson Method

Graph $G = (V, E, c)$:



Residual Graph $G_f = (V, E_f, c_f)$:



Introduction

Ford-Fulkerson

A Glimpse at the Max-Flow Min-Cut Theorem

Analysis of Ford-Fulkerson

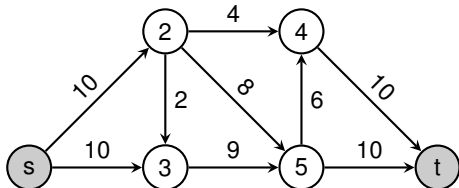


From Flows to Cuts

Cut

- A cut (S, T) is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$.

Graph $G = (V, E, c)$:

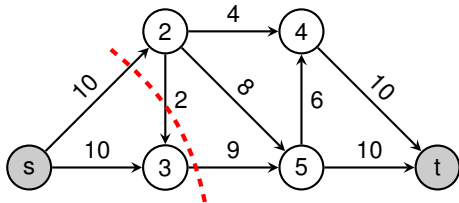


From Flows to Cuts

Cut

- A cut (S, T) is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$.

Graph $G = (V, E, c)$:



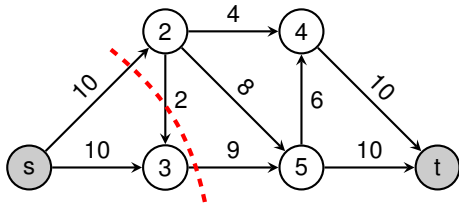
From Flows to Cuts

Cut

- A cut (S, T) is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$.
- The **capacity** of a cut (S, T) is the sum of capacities of the edges from S to T :

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) = \sum_{(u, v) \in E(S, T)} c(u, v)$$

Graph $G = (V, E, c)$:



$$c(\{s, 3\}, \{2, 4, 5, t\}) =$$



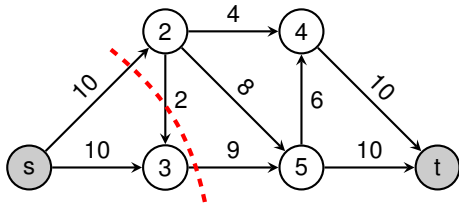
From Flows to Cuts

Cut

- A cut (S, T) is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$.
- The **capacity** of a cut (S, T) is the sum of capacities of the edges from S to T :

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) = \sum_{(u, v) \in E(S, T)} c(u, v)$$

Graph $G = (V, E, c)$:



$$c(\{s, 3\}, \{2, 4, 5, t\}) = 10 + 9 = 19$$



From Flows to Cuts

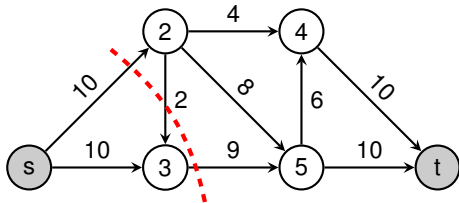
Cut

- A **cut** (S, T) is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$.
- The **capacity** of a cut (S, T) is the sum of capacities of the edges from S to T :

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) = \sum_{(u, v) \in E(S, T)} c(u, v)$$

- A **minimum cut** of a network is a cut whose capacity is minimum over all cuts of the network.

Graph $G = (V, E, c)$:



$$c(\{s, 3\}, \{2, 4, 5, t\}) = 10 + 9 = 19$$



From Flows to Cuts

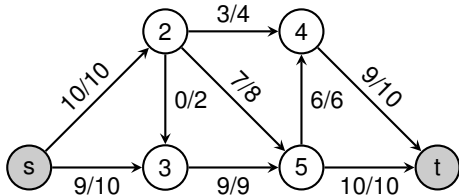
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



From Flows to Cuts

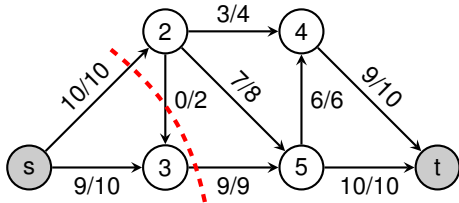
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



From Flows to Cuts

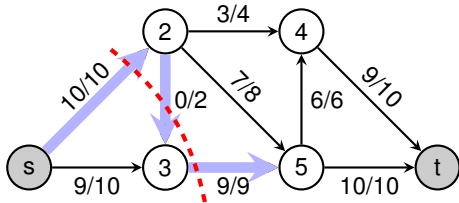
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



From Flows to Cuts

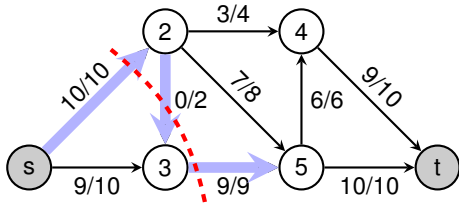
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



$$10 - 0 + 9 = 19$$



From Flows to Cuts

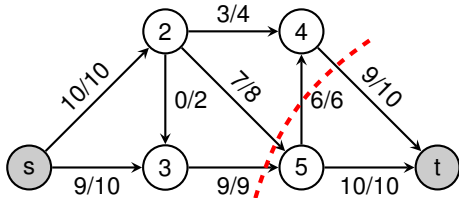
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



From Flows to Cuts

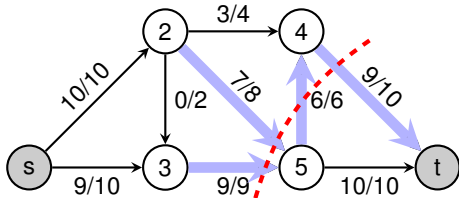
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



From Flows to Cuts

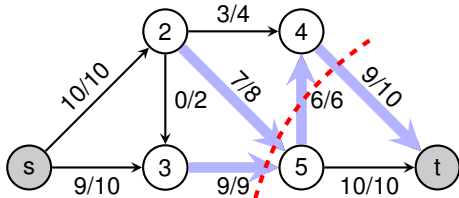
Theorem (Max-Flow Min-Cut Theorem)

The value of the max-flow is equal to the capacity of the min-cut, that is

$$\max_f |f| = \min_{S, T \subseteq V} c(S, T).$$

Graph $G = (V, E, c)$:

$|f| = 19$



$$9 + 7 - 6 + 9 = 19$$



Outline

Introduction

Ford-Fulkerson

A Glimpse at the Max-Flow Min-Cut Theorem

Analysis of Ford-Fulkerson



Analysis of Ford-Fulkerson

```
0: def FordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```



Analysis of Ford-Fulkerson

```
0: def FordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Lemma

If all capacities $c(u, v)$ are integral, then the flow at every iteration of Ford-Fulkerson is integral.



Analysis of Ford-Fulkerson

```
0: def FordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Lemma

If all capacities $c(u, v)$ are integral, then the flow at every iteration of Ford-Fulkerson is integral.

Flow before iteration integral
& capacities in G_f are integral
 \Rightarrow Flow after iteration integral



Analysis of Ford-Fulkerson

```
0: def FordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Lemma

If all capacities $c(u, v)$ are integral, then the flow at every iteration of Ford-Fulkerson is integral.

Theorem

For integral capacities $c(u, v)$, Ford-Fulkerson **terminates** after $C := \max_{u,v} c(u, v)$ iterations and returns the **maximum flow**.



Analysis of Ford-Fulkerson

```
0: def FordFulkerson(G)
1:   initialize flow to 0 on all edges
2:   while an augmenting path in  $G_f$  can be found:
3:     push as much extra flow as possible through it
```

Lemma

If all capacities $c(u, v)$ are integral, then the flow at every iteration of Ford-Fulkerson is integral.

Theorem

For integral capacities $c(u, v)$, Ford-Fulkerson **terminates** after $C := \max_{u,v} c(u, v)$ iterations and returns the **maximum flow**.

(proof omitted here, see CLRS3)

