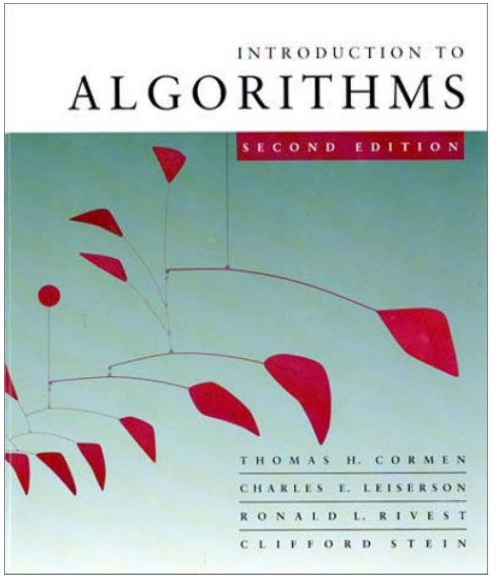
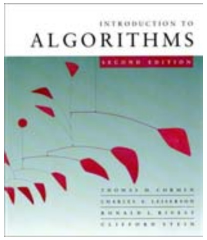
CS 5633 -- Spring 2008



Flow Networks

Carola Wenk

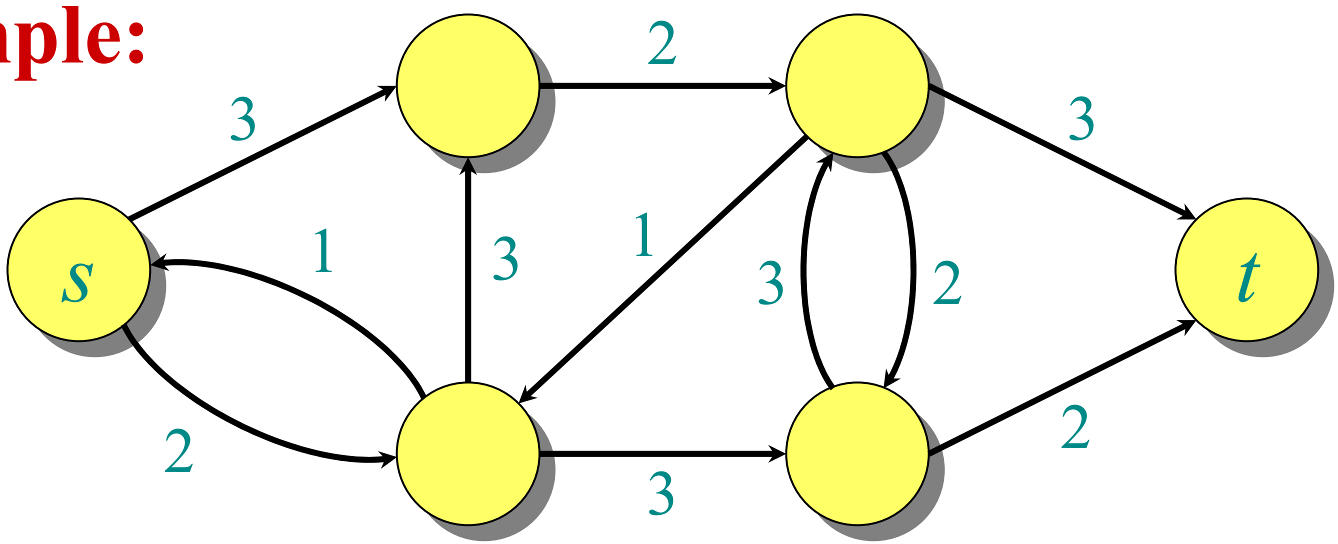
Slides courtesy of Charles Leiserson with small changes by Carola Wenk

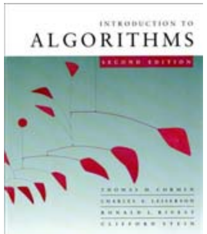


Flow networks

Definition. A ***flow network*** is a directed graph $G = (V, E)$ with two distinguished vertices: a ***source*** s and a ***sink*** t. Each edge $(u, v) \in E$ has a nonnegative ***capacity*** $c(u, v)$. If $(u, v) \notin E$, then $c(u, v) = 0$.

Example:





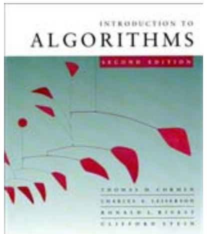
Flow networks

Definition. A ***positive flow*** on G is a function $p : V \times V \to \mathbb{R}$ satisfying the following:

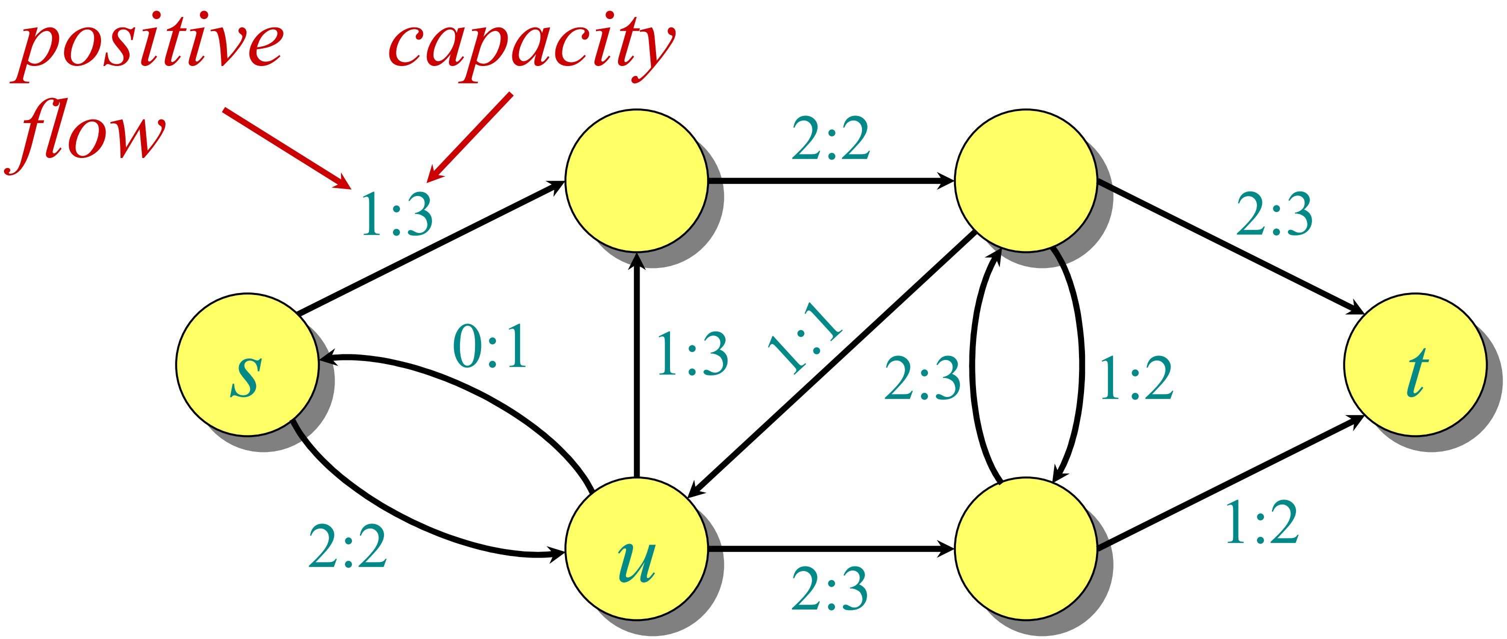
- ***Capacity constraint:*** For all $u, v \in V$,
$$0 \le p(u, v) \le c(u, v).$$
- ***Flow conservation:*** For all $u \in V \setminus \{s, t\}$,
$$\sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u) = 0 .$$

The ***value*** of a flow is the net flow out of the source:

$$\sum_{v \in V} p(s,v) - \sum_{v \in V} p(v,s) .$$



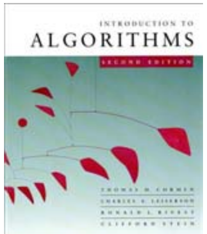
A flow on a network



Flow conservation (like Kirchoff's current law):

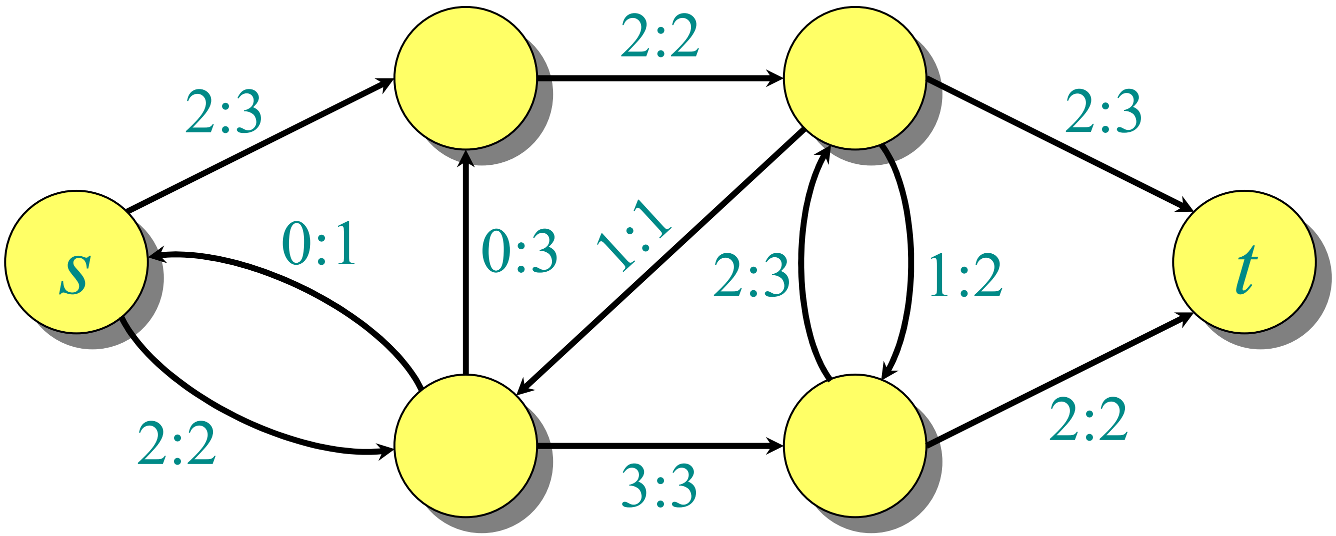
- Flow into u is $2 + 1 = 3$.
- Flow out of u is $0 + 1 + 2 = 3$.

The value of this flow is $1 - 0 + 2 = 3$.



The maximum-flow problem

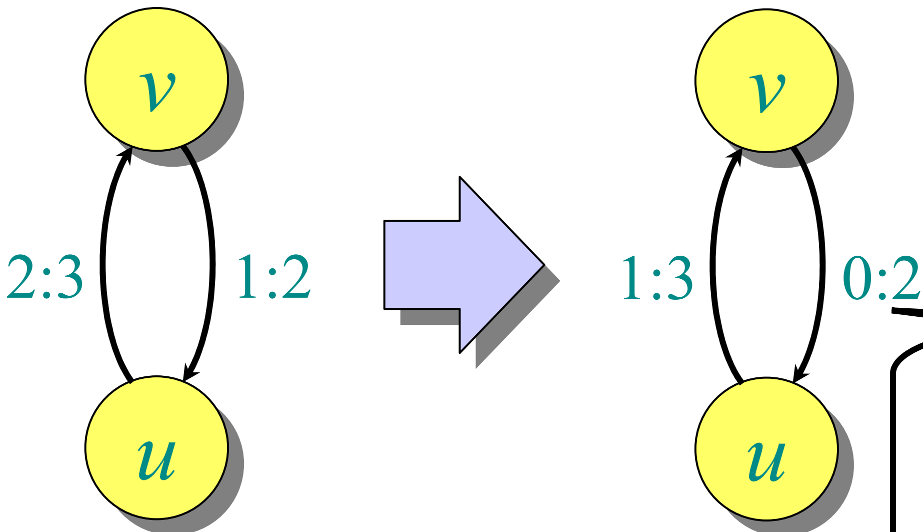
Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



The value of the maximum flow is 4.

Flow cancellation

Without loss of generality, positive flow goes either from *u* to *v*, or from *v* to *u*, but not both.

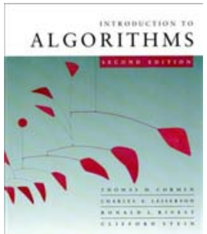


Net flow from *u* to *v* in both cases is 1.

On the following slides the (net) flow on this edge will be the negated flow of the other direction, so, -1.

The capacity constraint and flow conservation are preserved by this transformation.

INTUITION: View flow as a *rate*, not a *quantity*.

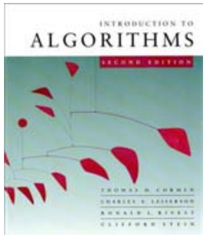


A notational simplification

IDEA: Work with the net flow between two vertices, rather than with the positive flow.

Definition. A ***(net) flow*** on G is a function $f : V \times V \to \mathbb{R}$ satisfying the following:

- ***Capacity constraint:*** For all $u, v \in V$,
$$f(u, v) \le c(u, v).$$
- ***Flow conservation:*** For all $u \in V \setminus \{s, t\}$,
$$\sum_{v \in V} f(u, v) = 0.$$ ← *One summation instead of two.*
- ***Skew symmetry:*** For all $u, v \in V$,
$$f(u, v) = -f(v, u).$$



Equivalence of definitions

Theorem. The two definitions are equivalent.

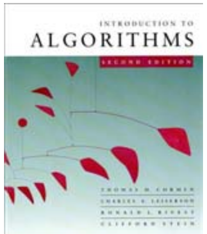
Proof. ($\Rightarrow$) Let $f(u, v) = p(u, v) - p(v, u)$.

- ***Capacity constraint:*** Since $p(u, v) \le c(u, v)$ and $p(v, u) \ge 0$, we have $f(u, v) \le c(u, v)$.
- ***Flow conservation:***

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} \left(p(u,v) - p(v,u)\right)$$
$$= \sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u)$$

- ***Skew symmetry:***

$$f(u, v) = p(u, v) - p(v, u)$$
$$= -(p(v, u) - p(u, v))$$
$$= -f(v, u).$$



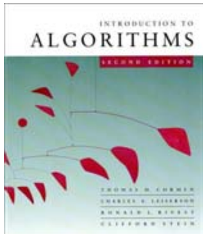
Proof (continued)

(⇐) Let

$$p(u, v) = \begin{cases} f(u, v) & \text{if } f(u, v) > 0, \\ 0 & \text{if } f(u, v) \le 0. \end{cases}$$

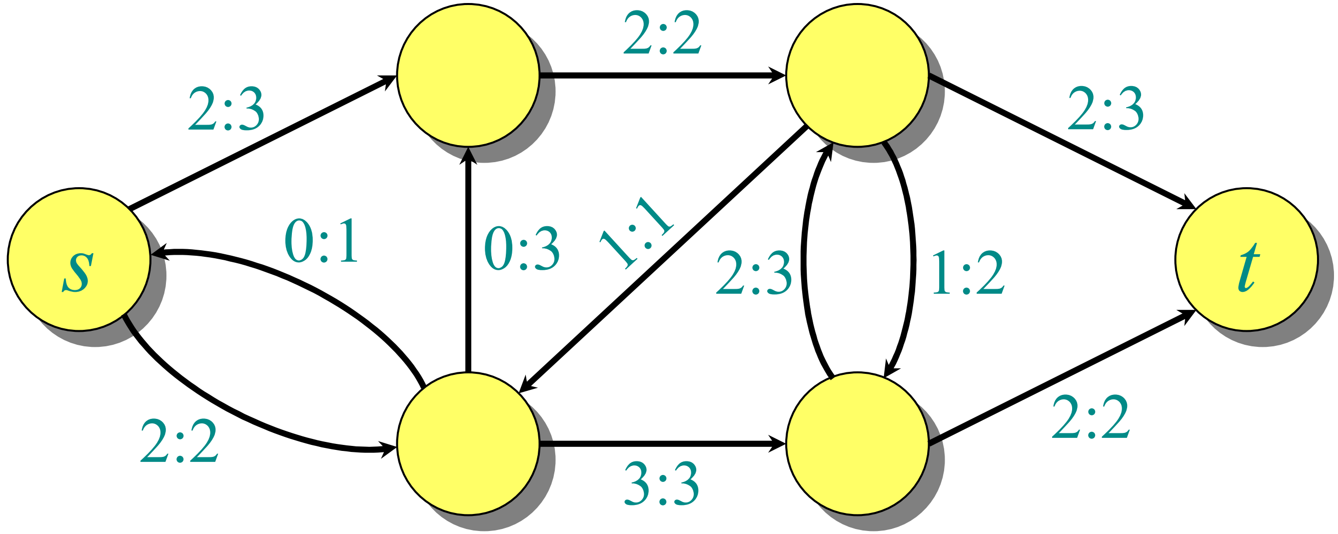
- ***Capacity constraint:*** By definition, $p(u, v) \ge 0$. Since $f(u, v) \le c(u, v)$, it follows that $p(u, v) \le c(u, v)$.
- ***Flow conservation:*** If $f(u, v) > 0$, then $p(u, v) - p(v, u) = f(u, v)$. If $f(u, v) \le 0$, then $p(u, v) - p(v, u) = -f(v, u) = f(u, v)$ by skew symmetry. Therefore,

$$\sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u) = \sum_{v \in V} f(u,v). \quad \square$$

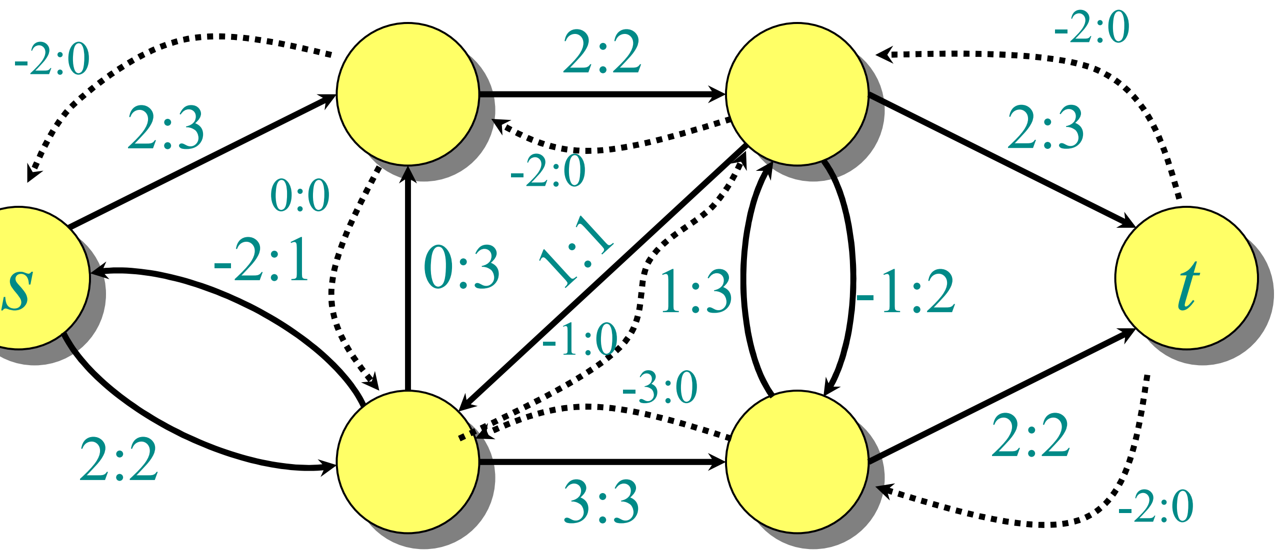


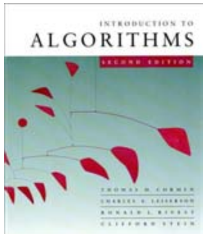
Positive flow vs. (net) flow

Positive flow:



(Net) flow:





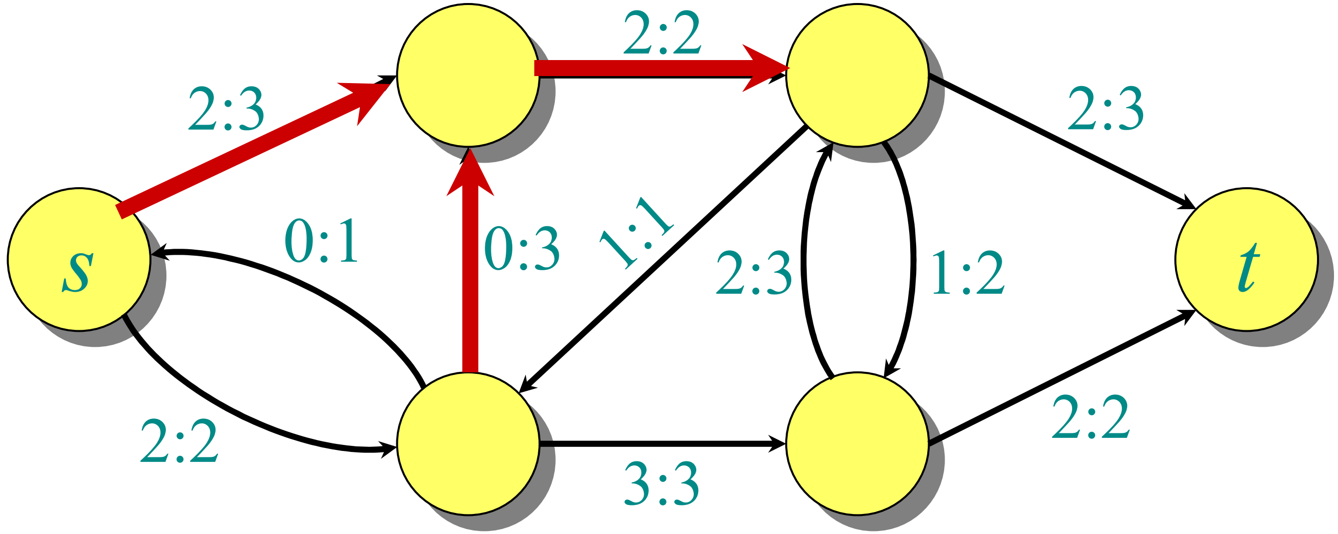
Positive flow vs. (net) flow

Positive flow:

Flow conserv.:

2+0 - 2 = 0

(2+0: in-coming; 2: outgoing)

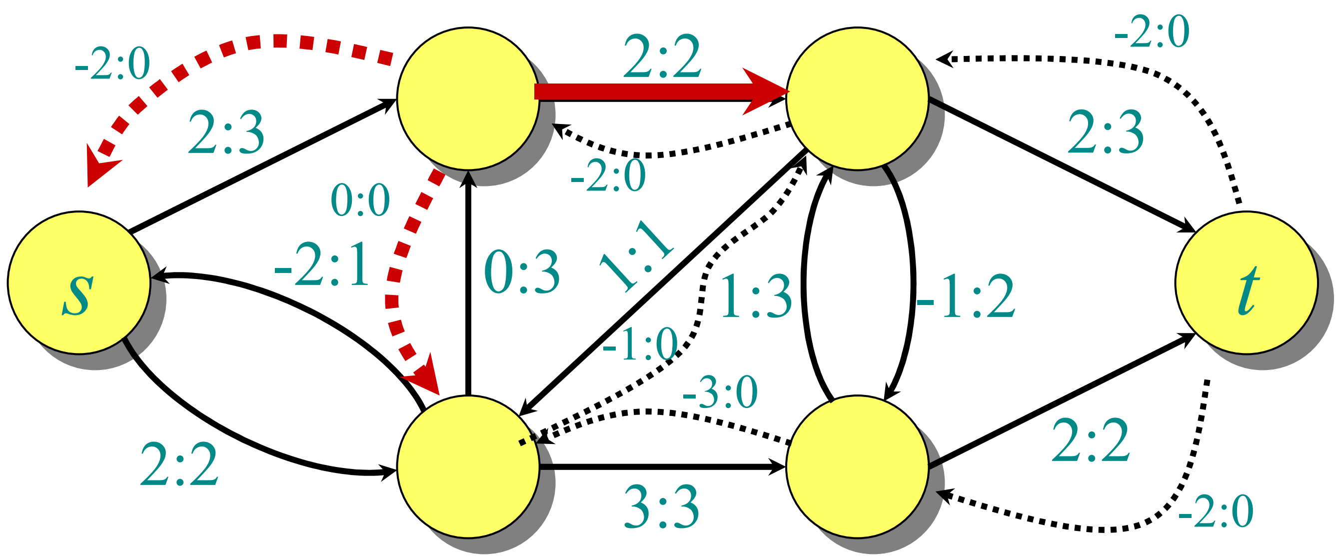


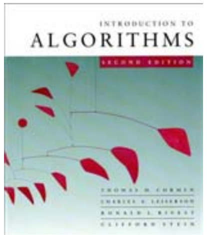
(Net) flow:

Flow conserv.:

-2-0 + 2 = 0

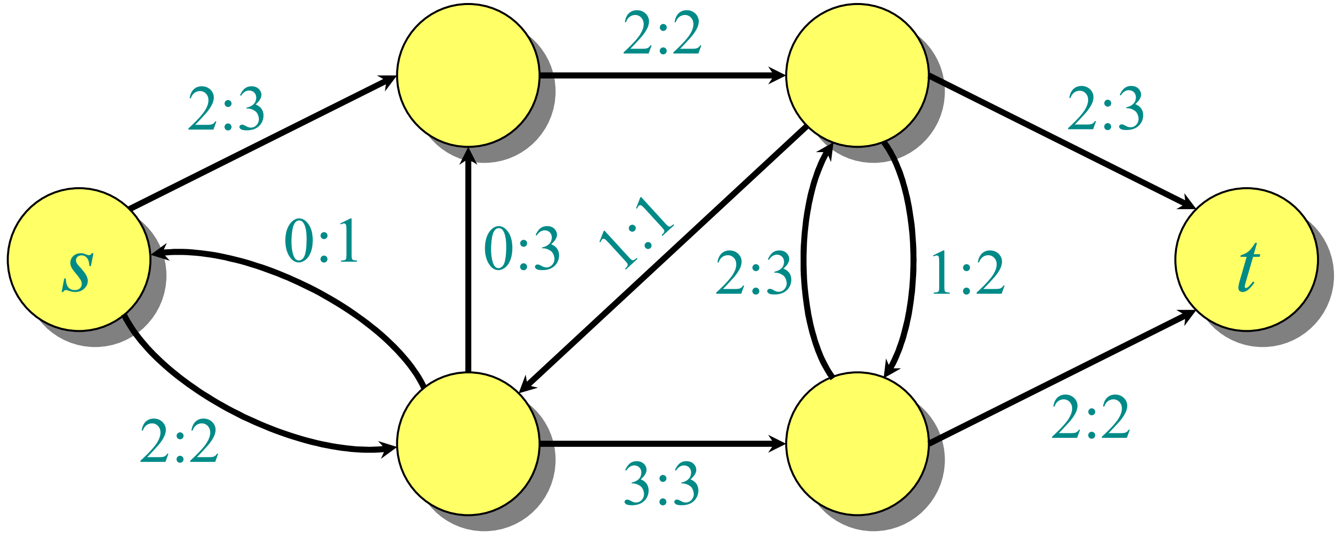
(-2-0 + 2: outgoing)



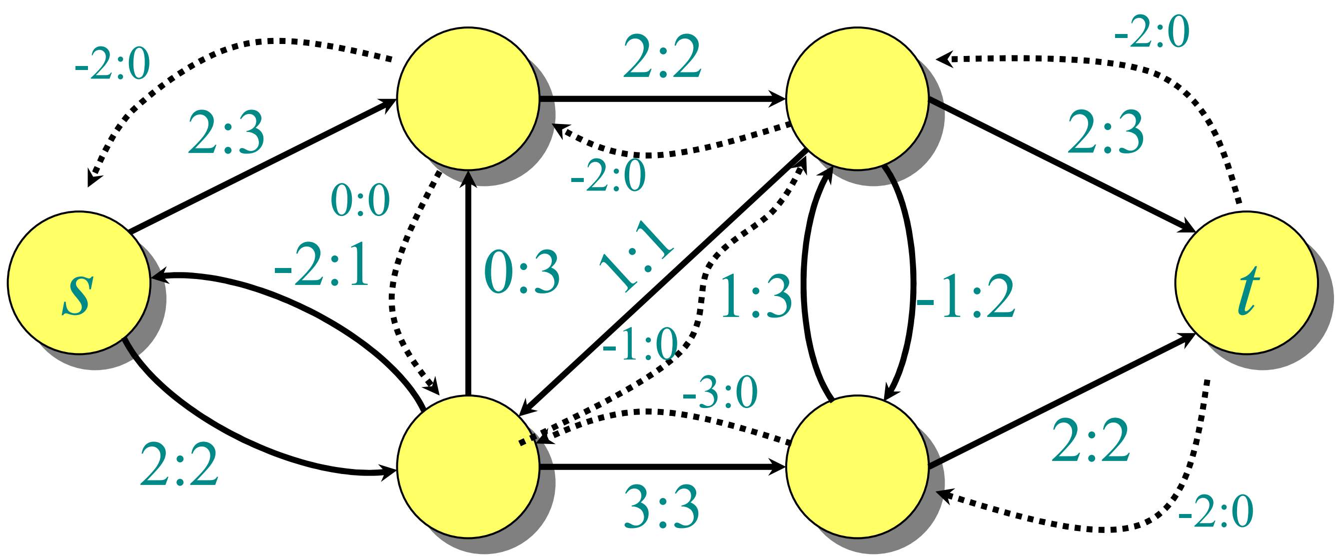


Positive flow vs. (net) flow

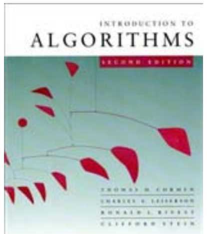
Positive flow:



(Net) flow:



Edges with 0-capacity are usually omitted, even if they carry a negative flow!



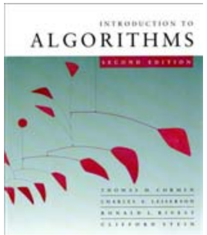
Notation

Definition. The ***value*** of a flow f, denoted by $|f|$, is given by

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

Implicit summation notation: A set used in an arithmetic formula represents a sum over the elements of the set.

- **Example** — flow conservation: $f(u, V) = 0$ for all $u \in V \setminus \{s, t\}$.



Simple properties of flow

Lemma.

1. $f(X, X) = 0$,

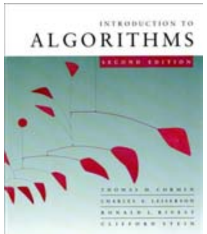
2. $f(X, Y) = -f(Y, X)$,

3. $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset$. □

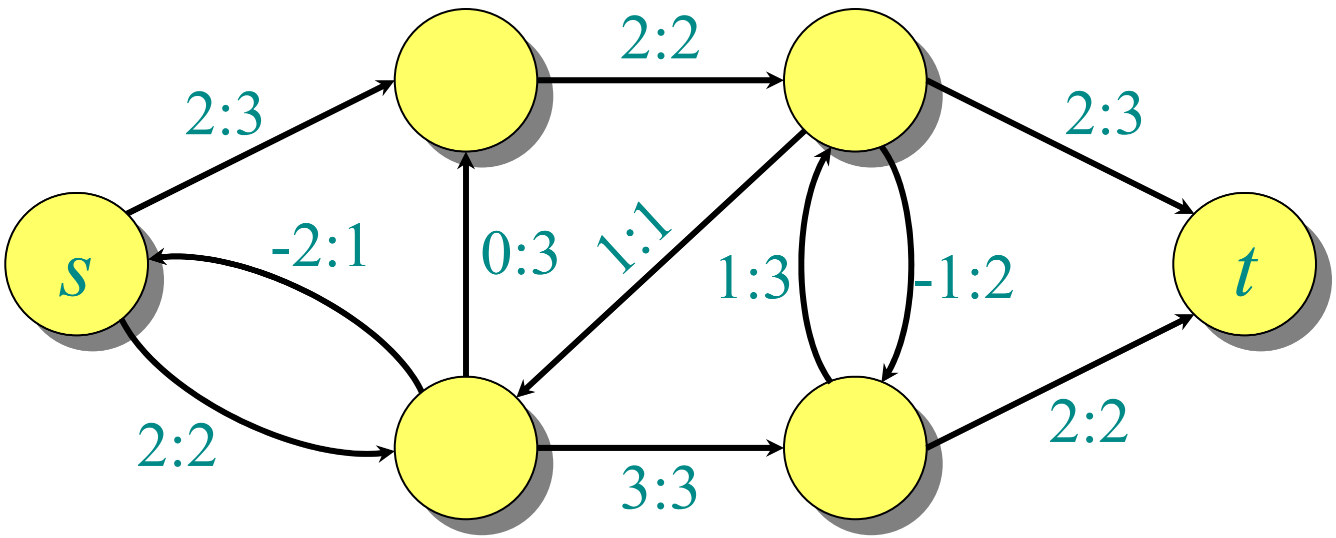
Theorem. $|f| = f(V, t)$.

Proof.

$$
\begin{aligned}
|f| &= f(s, V) && \textit{3.} \\
&= f(V, V) - f(V \setminus \{s\}, V) && \textit{1., 2.} \\
&= f(V, V \setminus \{s\}) && \textit{2., 3.} \\
&= f(V, t) + f(V, V \setminus \{s,t\}) && \textit{Flow conservation} \\
&= f(V, t).
\end{aligned}
$$
□

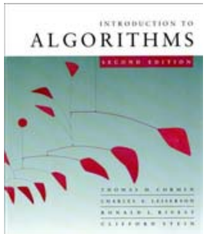


Flow into the sink



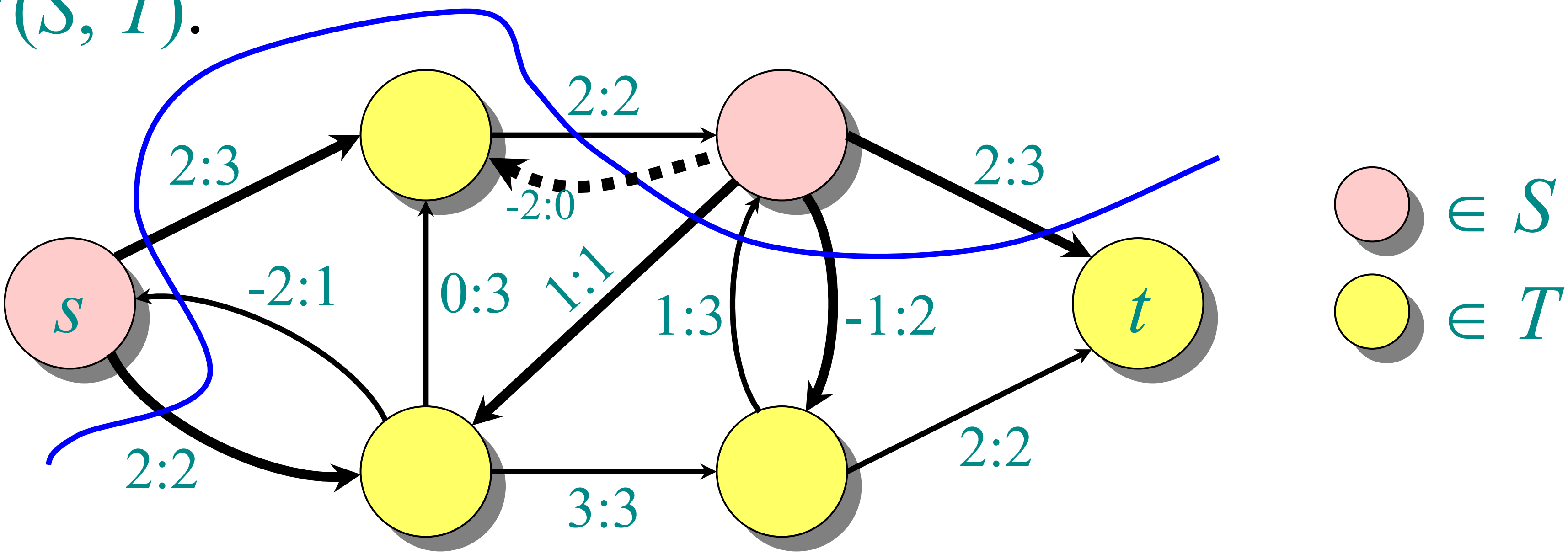
$|f| = f(s, V) = 4$

$f(V, t) = 4$

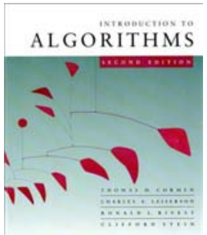


Cuts

Definition. A ***cut*** (S, T) of a flow network $G = (V, E)$ is a partition of V such that $s \in S$ and $t \in T$. If f is a flow on G, then the ***flow across the cut*** is $f(S, T)$.



$$f(S, T) = (2 + 2) + (-2 + 1 - 1 + 2)$$
$$= 4$$

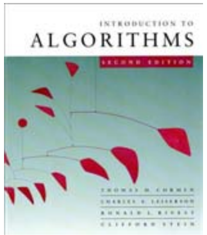


Another characterization of flow value

Lemma. For any flow f and any cut (S, T), we have $|f| = f(S, T)$.

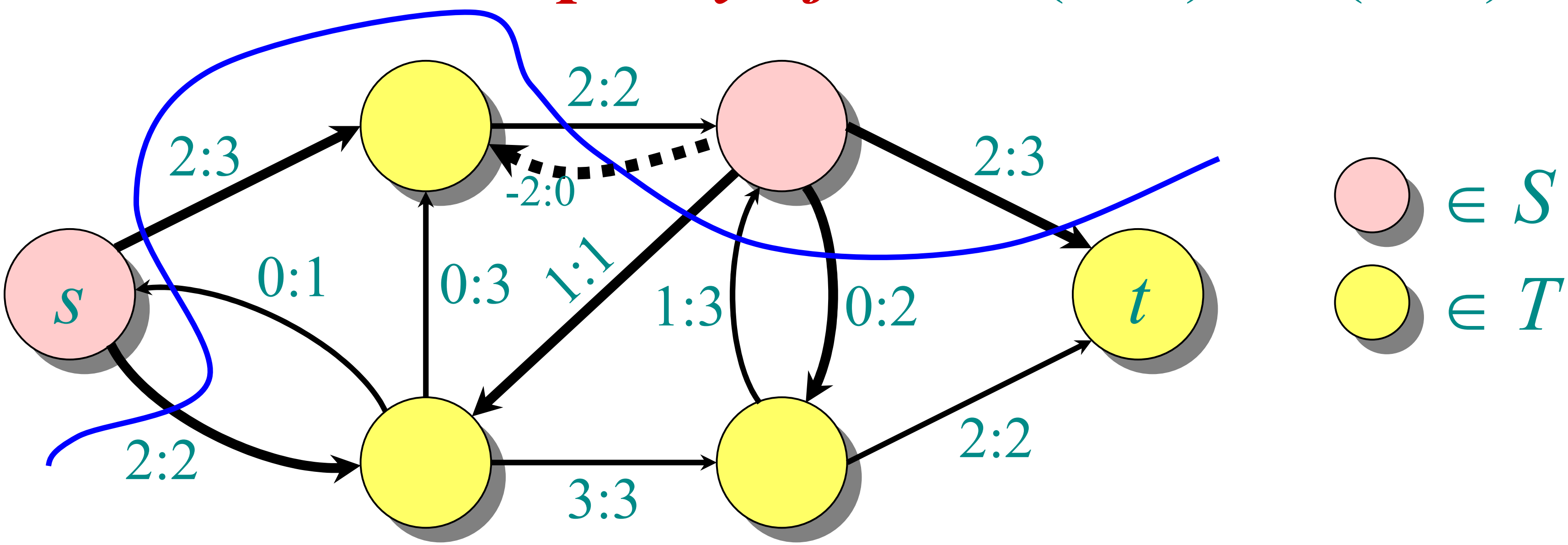
Proof.

$$\begin{aligned} f(S, T) &= f(S, V) - f(S, S) \\ &= f(S, V) \\ &= f(s, V) + f(S \backslash \{s\}, V) \\ &= f(s, V) \\ &= |f|. \quad \square \end{aligned}$$

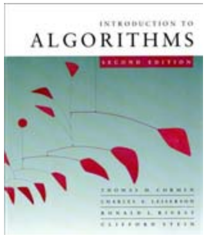


Capacity of a cut

Definition. The ***capacity of a cut*** (S, T) is $c(S, T)$.



$$c(S, T) = (2 + 3) + (0 + 1 + 2 + 3)$$
$$= 11$$

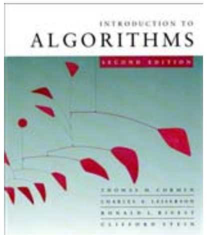


Upper bound on the maximum flow value

Theorem. The value of any flow is bounded from above by the capacity of any cut: $|f| \le c(S,T)$.

Proof.

$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u \in S} \sum_{v \in T} f(u,v) \\
&\le \sum_{u \in S} \sum_{v \in T} c(u,v) \\
&= c(S,T) \qquad \blacksquare
\end{aligned}
$$



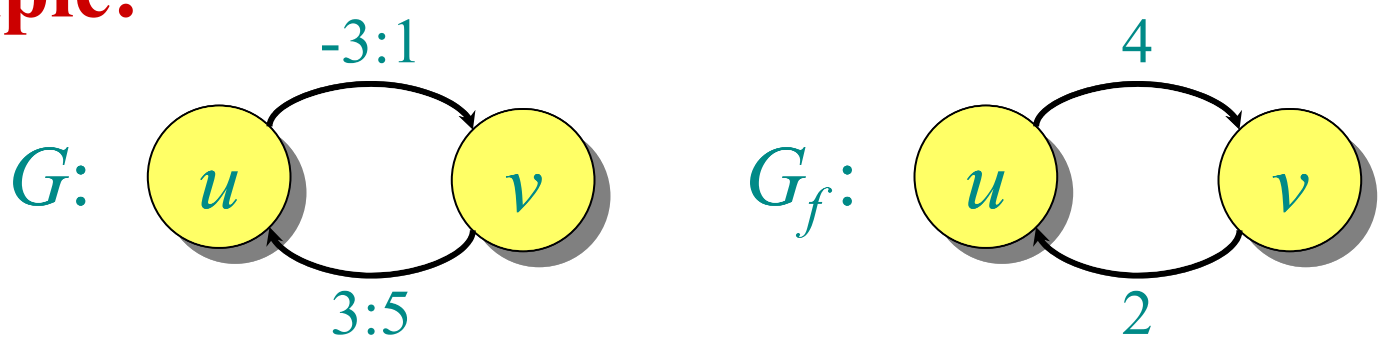
Residual network

Definition. Let f be a flow on $G = (V, E)$. The ***residual network*** $G_f(V, E_f)$ is the graph with strictly positive ***residual capacities***

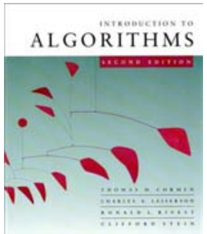
$$c_f(u, v) = c(u, v) - f(u, v) > 0.$$

Edges in E_f admit more flow.

Example:



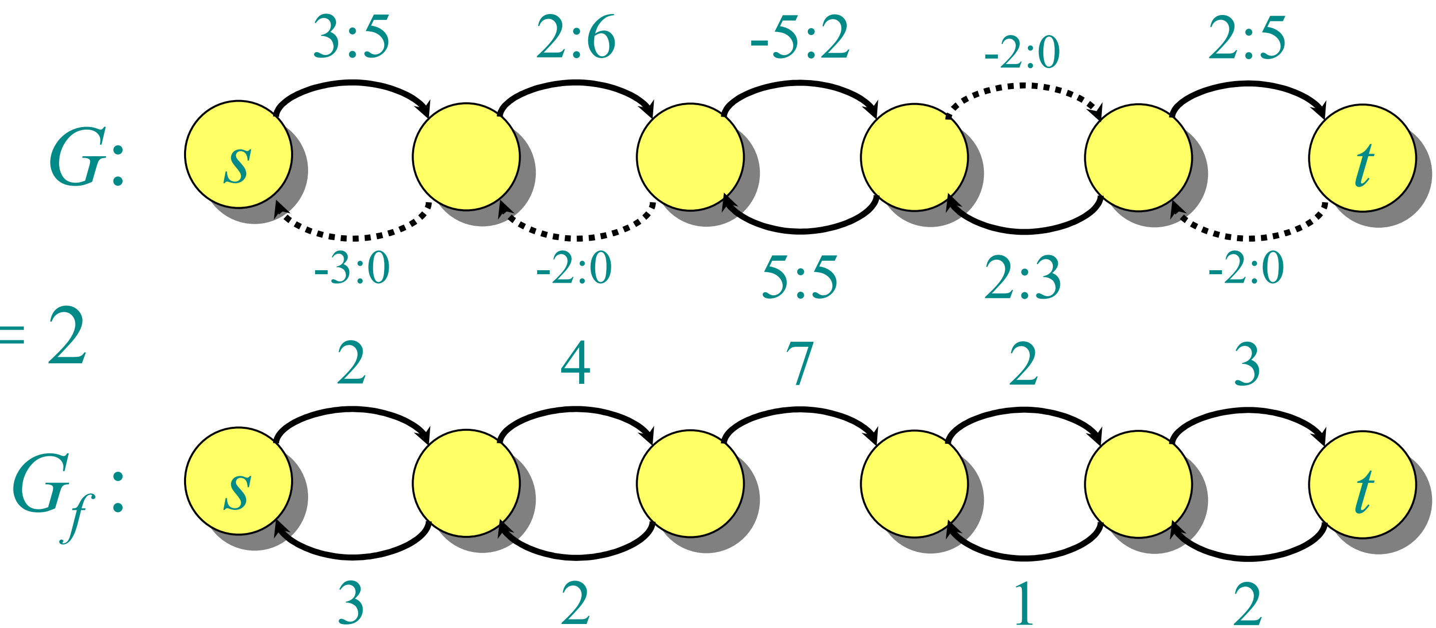
Lemma. $|E_f| \leq 2|E|$. ◻



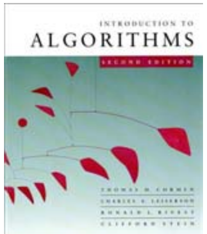
Augmenting paths

Definition. Any path from s to t in G_f is an ***augmenting path*** in G with respect to f. The flow value can be increased along an augmenting path p by $c_f(p) = \min_{(u,v)\in p} \{c_f(u,v)\}$.

Ex.:



$c_f(p) = 2$



Max-flow, min-cut theorem

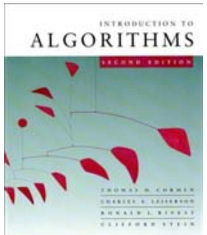
Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T). ← min-cut
2. f is a maximum flow.
3. f admits no augmenting paths.

Proof.

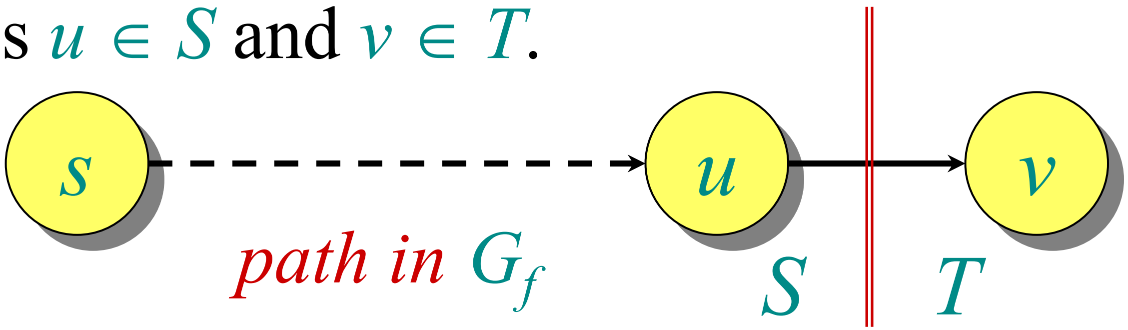
(1) ⇒ (2): Since $|f| \leq c(S, T)$ for any cut (S, T) (by the theorem from 3 slides back), the assumption that $|f| = c(S, T)$ implies that f is a maximum flow.

(2) ⇒ (3): If there was an augmenting path, the flow value could be increased, contradicting the maximality of f.

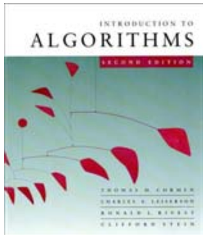


Proof (continued)

(*3*) ⇒ (*1*): Define $S = \{v \in V :$ there exists a path in G_f from s to $v\}$, and let $T = V \setminus S$. Since f admits no augmenting paths, there is no path from s to t in G_f. Hence, $s \in S$ and $t \in T$, and thus (S, T) is a cut. Consider any vertices $u \in S$ and $v \in T$.



We must have $c_f(u, v) = 0$, since if $c_f(u, v) > 0$, then $v \in S$, not $v \in T$ as assumed. Thus, $f(u, v) = c(u, v)$, since $c_f(u, v) = c(u, v) - f(u, v)$. Summing over all $u \in S$ and $v \in T$ yields $f(S, T) = c(S, T)$, and since $|f| = f(S, T)$, the theorem follows. □



Ford-Fulkerson max-flow algorithm

Algorithm:

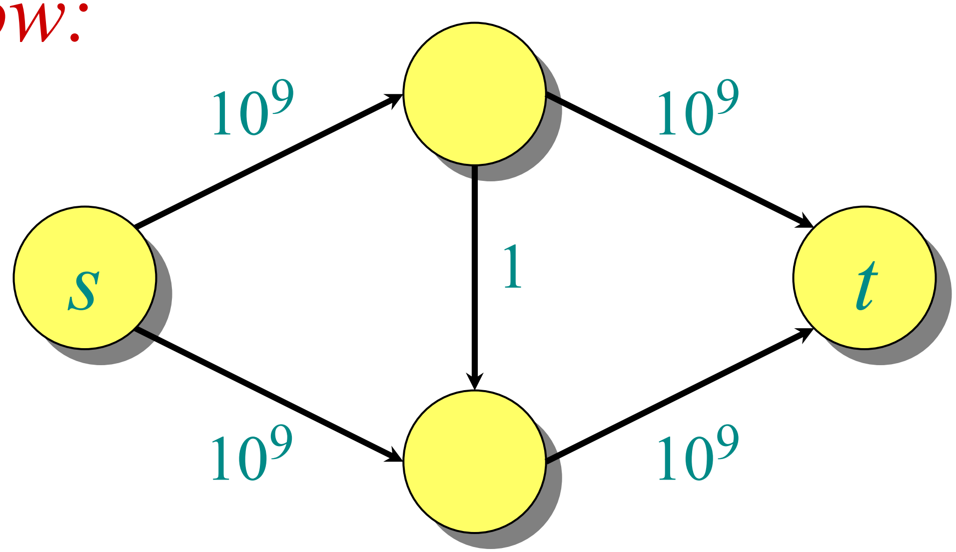
$f[u, v] \leftarrow 0$ for all $u, v \in V$

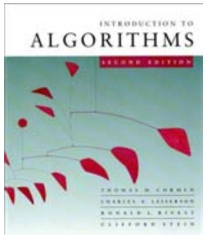
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

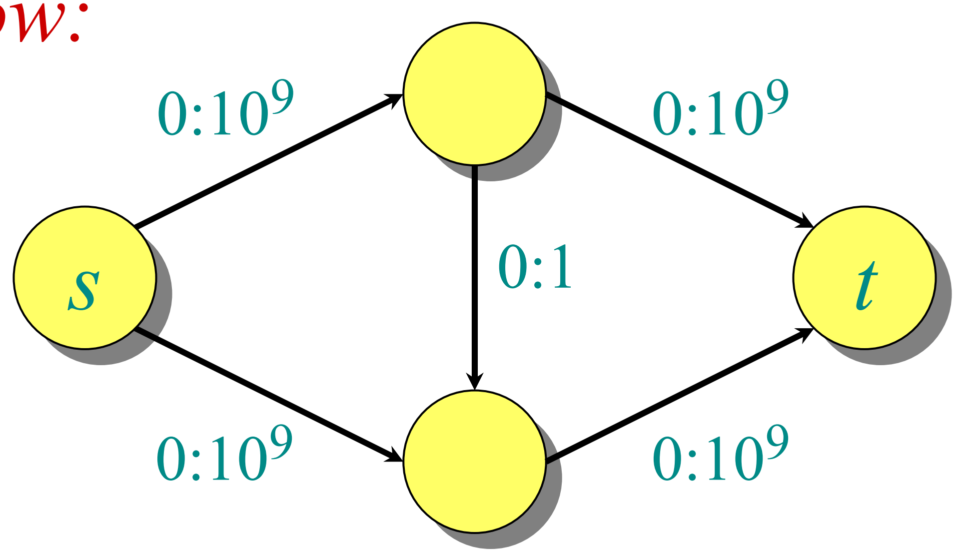
$f[u, v] \leftarrow 0$ for all $u, v \in V$

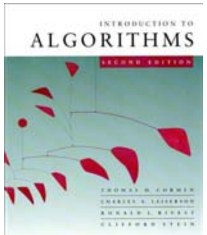
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

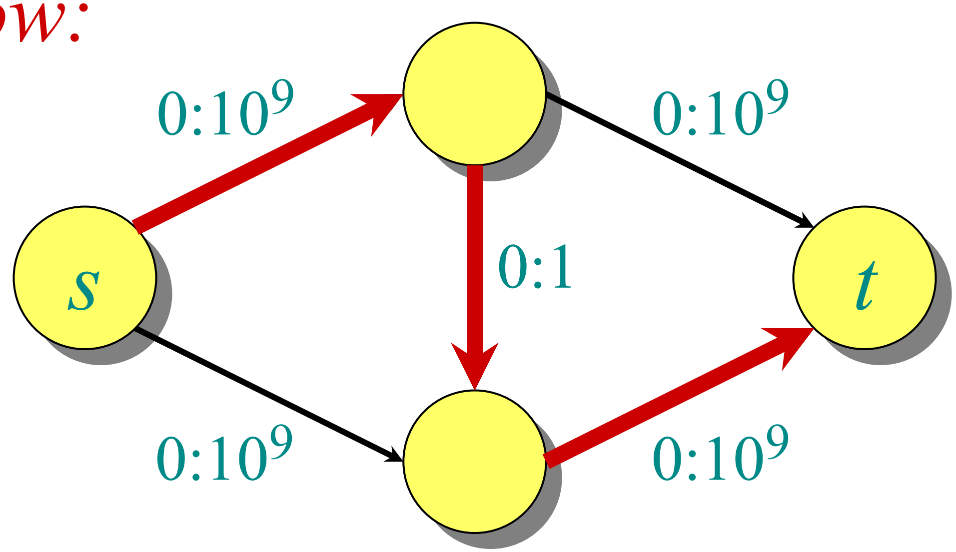
$f[u, v] \leftarrow 0$ for all $u, v \in V$

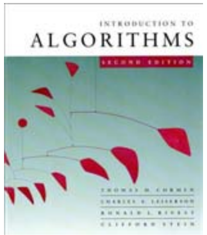
while an augmenting path p in G wrt f exists

  do augment f by $c_f(p)$

Can be slow:

G:





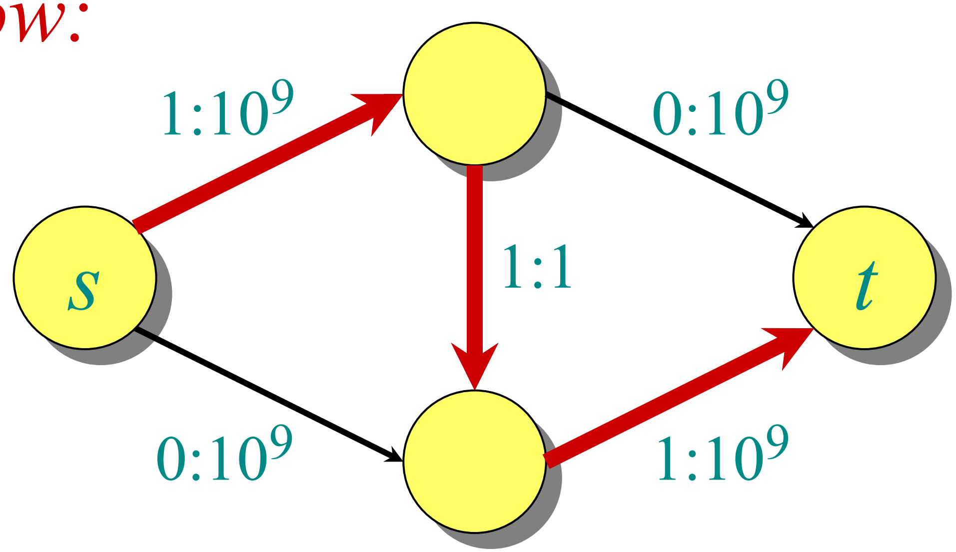
Ford-Fulkerson max-flow algorithm

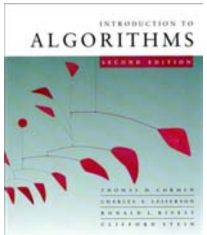
Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
  do augment f by $c_f(p)$

Can be slow:

G:





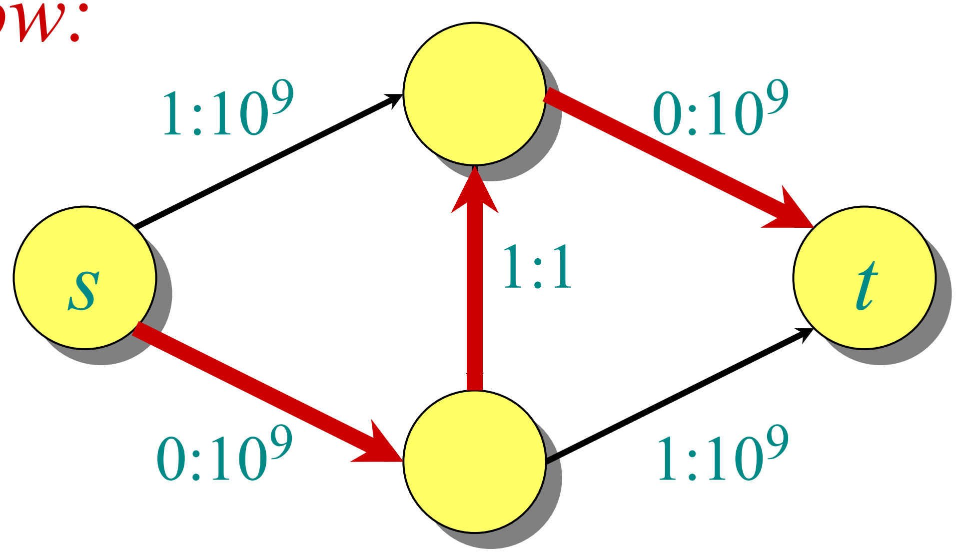
Ford-Fulkerson max-flow algorithm

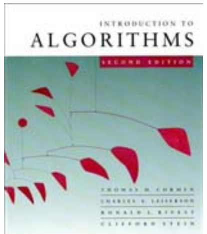
Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

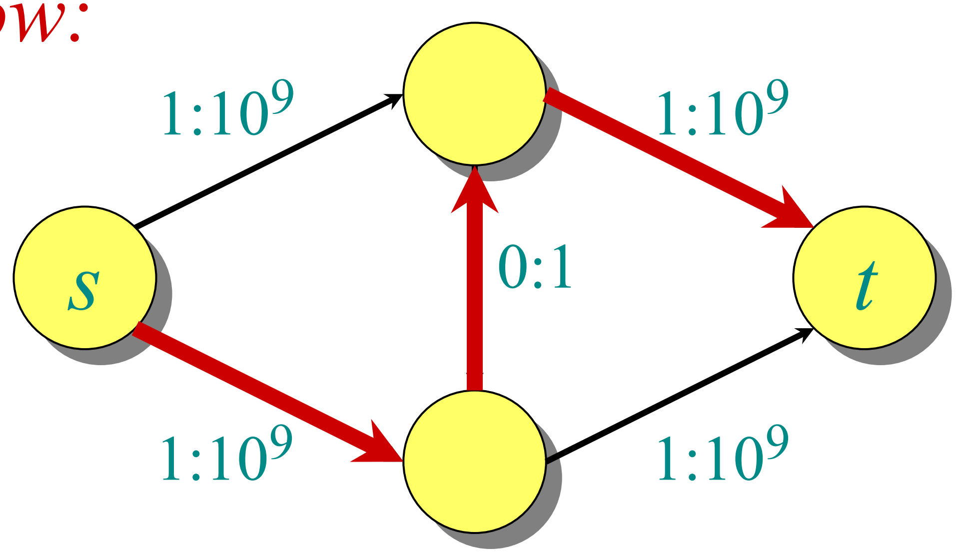
$f[u, v] \leftarrow 0$ for all $u, v \in V$

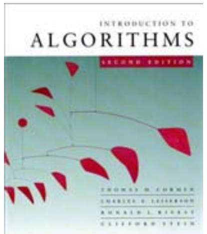
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

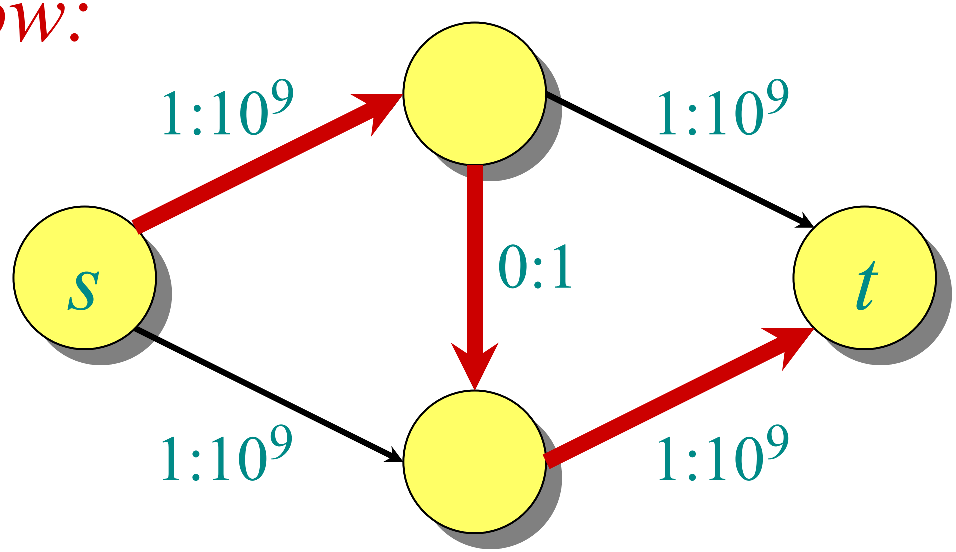
$f[u, v] \leftarrow 0$ for all $u, v \in V$

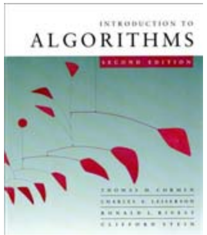
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





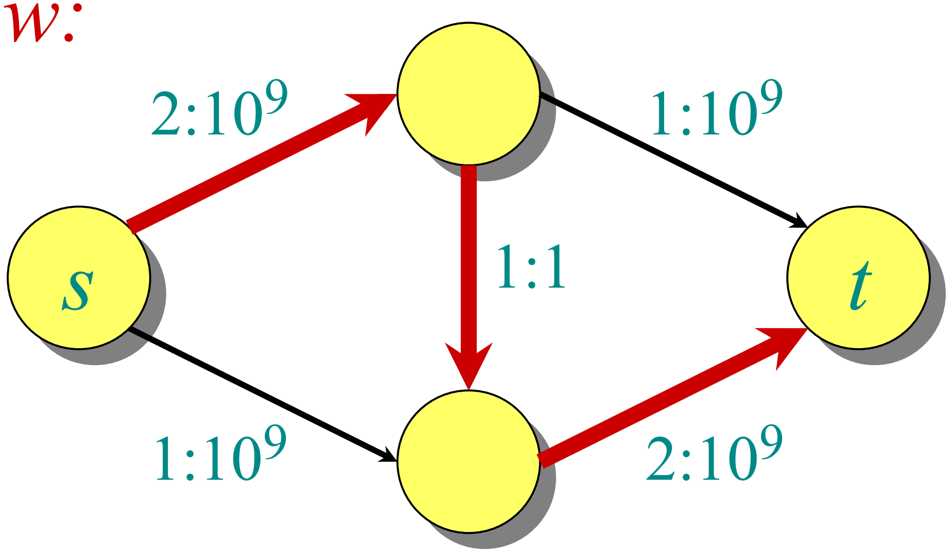
Ford-Fulkerson max-flow algorithm

Algorithm:

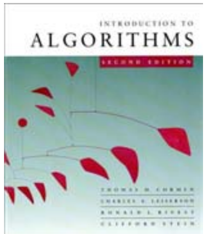
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
    do augment f by $c_f(p)$

Can be slow:

G:



2 billion iterations on a graph with 4 vertices!



Ford-Fulkerson max-flow algorithm

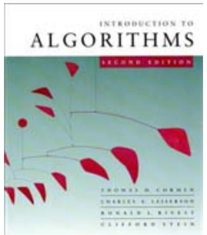
Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
do augment f by $c_f(p)$

Runtime:

- Let $|f^*|$ be the value of a maximum flow, and assume it is an integral value.
- The initialization takes $O(|E|)$ time
- There are at most $|f^*|$ iterations of the loop
- Find an augmenting path with DFS in $O(|V|+|E|)$ time
- Each augmentation takes $O(|V|)$ time

$\Rightarrow O(|E| \cdot |f^*|)$ time in total

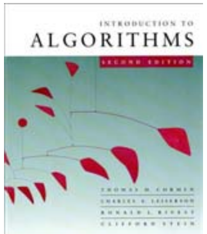


Edmonds-Karp algorithm

Edmonds and Karp noticed that many people's implementations of Ford-Fulkerson augment along a ***breadth-first augmenting path***: a shortest path in G_f from s to t where each edge has weight 1. These implementations would always run relatively fast.

Since a breadth-first augmenting path can be found in $O(V+E)$ time, their analysis, which provided the first polynomial-time bound on maximum flow, focuses on bounding the number of flow augmentations.

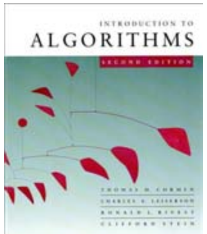
(In independent work, Dinic also gave polynomial-time bounds.)



Running time of Edmonds-Karp

- One can show that the number of flow augmentations (i.e., the number of iterations of the while loop) is $O(V\,E)$.
- Breadth-first search runs in $O(V+E)$ time
- All other bookkeeping is $O(V)$ per augmentation.

$\Rightarrow$ The Edmonds-Karp maximum-flow algorithm runs in $O(V\,E^2)$ time.

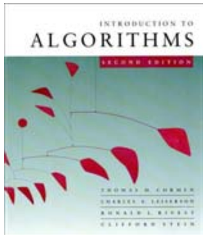


Monotonicity lemma

Lemma. Let $\delta(v) = \delta_f(s, v)$ be the breadth-first distance from s to v in G_f. During the Edmonds-Karp algorithm, $\delta(v)$ increases monotonically.

Proof. Suppose that f is a flow on G, and augmentation produces a new flow f'. Let $\delta'(v) = \delta_{f'}(s, v)$. We'll show that $\delta'(v) \geq \delta(v)$ by induction on $\delta(v)$. For the base case, $\delta'(s) = \delta(s) = 0$.

For the inductive case, consider a breadth-first path $s \to \cdots \to u \to v$ in $G_{f'}$. We must have $\delta'(v) = \delta'(u) + 1$, since subpaths of shortest paths are shortest paths. Certainly, $(u, v) \in E_{f'}$, and now consider two cases depending on whether $(u, v) \in E_f$.



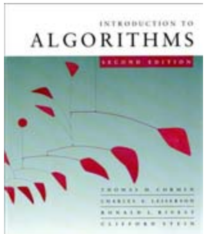
Case 1

Case: $(u, v) \in E_f$.

We have

$$\begin{aligned} \delta(v) &\le \delta(u) + 1 && \text{(triangle inequality)} \\ &\le \delta'(u) + 1 && \text{(induction)} \\ &= \delta'(v) && \text{(breadth-first path),} \end{aligned}$$

and thus monotonicity of $\delta(v)$ is established.



Case 2

Case: $(u, v) \notin E_f$.

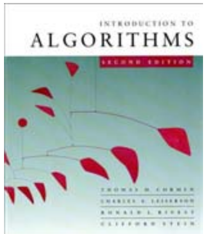
Since $(u, v) \in E_{f'}$, the augmenting path p that produced f' from f must have included (v, u). Moreover, p is a breadth-first path in G_f:

$$p = s \to \cdots \to v \to u \to \cdots \to t\,.$$

Thus, we have

$$\begin{aligned} \delta(v) &= \delta(u) - 1 && \text{(breadth-first path)} \\ &\le \delta'(u) - 1 && \text{(induction)} \\ &= \delta'(v) - 2 && \text{(breadth-first path)} \\ &< \delta'(v)\,, \end{aligned}$$

thereby establishing monotonicity for this case, too. ◻



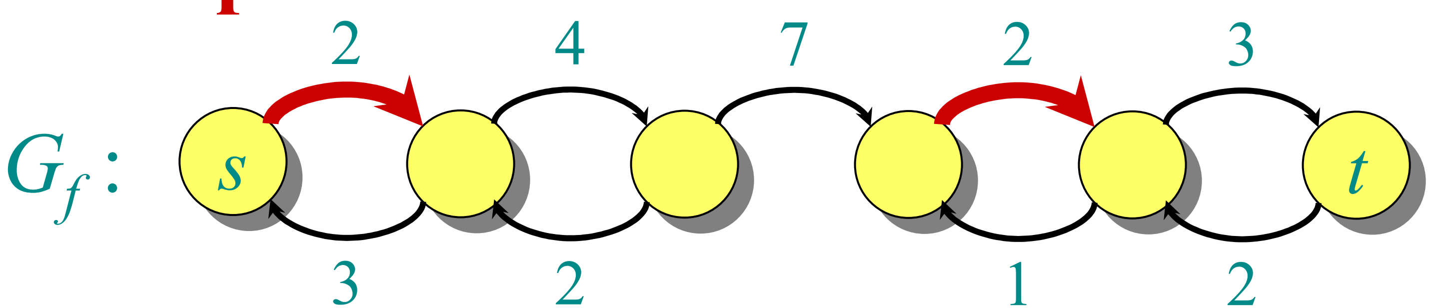
Counting flow augmentations

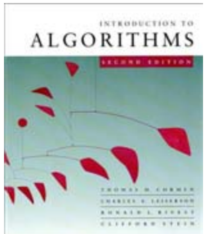
Theorem. The number of flow augmentations in the Edmonds-Karp algorithm (Ford-Fulkerson with breadth-first augmenting paths) is $O(VE)$.

Proof. Let p be an augmenting path, and suppose that we have $c_f(u, v) = c_f(p)$ for edge $(u, v) \in p$. Then, we say that (u, v) is ***critical***, and it disappears from the residual graph after flow augmentation.

Example:

$c_f(p) = 2$



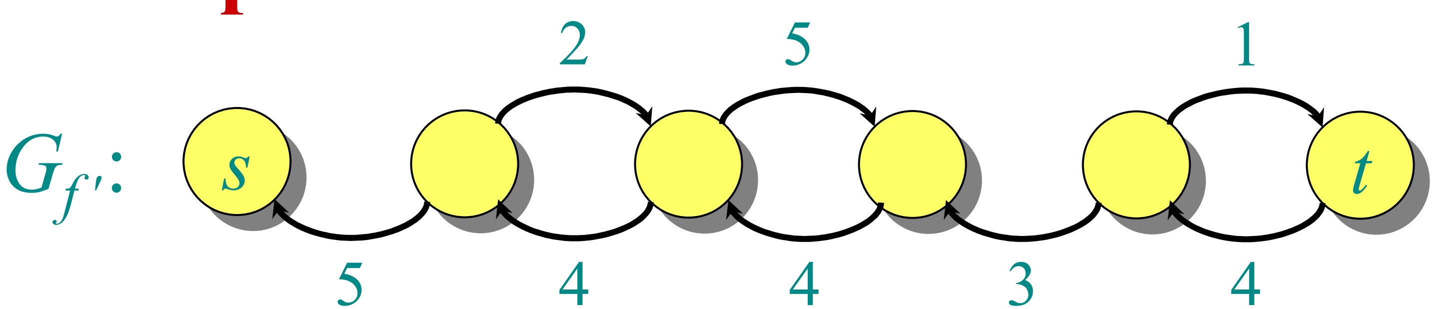


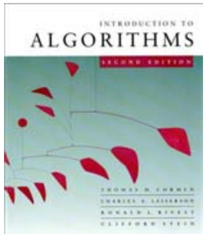
Counting flow augmentations

Theorem. The number of flow augmentations in the Edmonds-Karp algorithm (Ford-Fulkerson with breadth-first augmenting paths) is $O(VE)$.

Proof. Let p be an augmenting path, and suppose that the residual capacity of edge $(u, v) \in p$ is $c_f(u, v) = c_f(p)$. Then, we say (u, v) is ***critical***, and it disappears from the residual graph after flow augmentation.

Example:



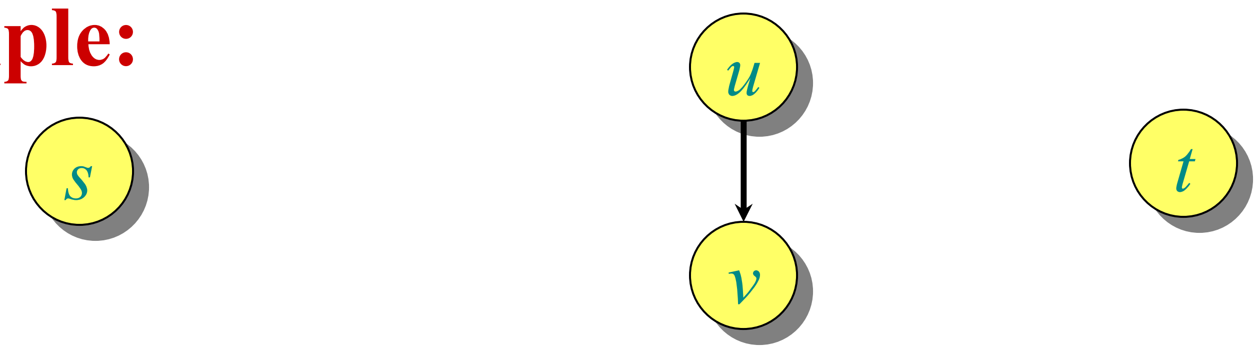


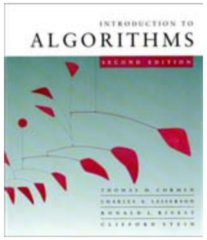
Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned} \delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)} \\ &\geq \delta(v) + 1 && \text{(monotonicity)} \\ &= \delta(u) + 2 && \text{(breadth-first path).} \end{aligned}$$

Example:



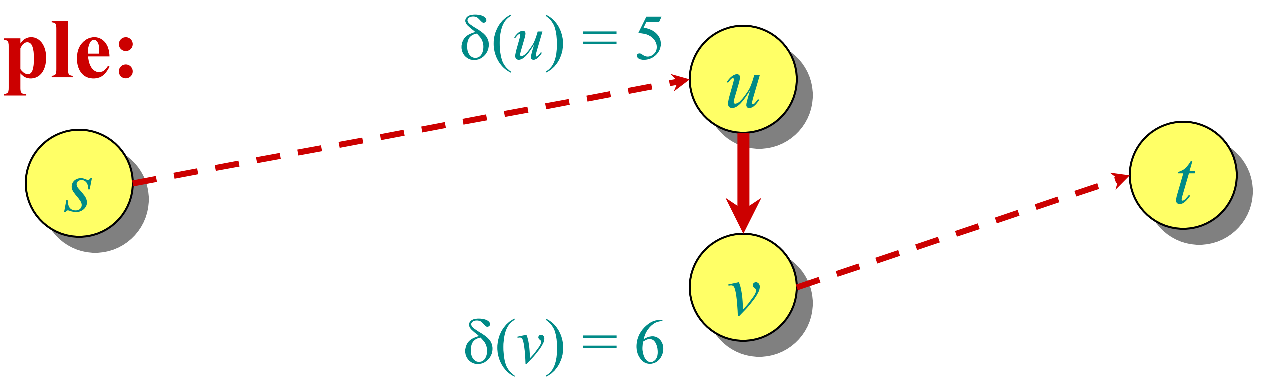


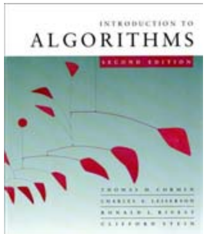
Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned}\delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)}\\ &\geq \delta(v) + 1 && \text{(monotonicity)}\\ &= \delta(u) + 2 && \text{(breadth-first path).}\end{aligned}$$

Example:



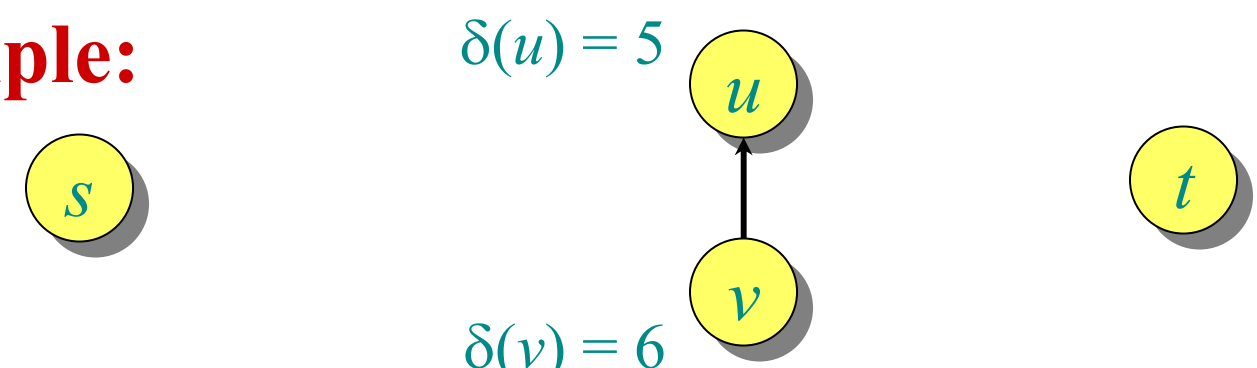


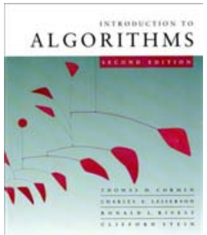
Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned} \delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)} \\ &\geq \delta(v) + 1 && \text{(monotonicity)} \\ &= \delta(u) + 2 && \text{(breadth-first path).} \end{aligned}$$

Example:



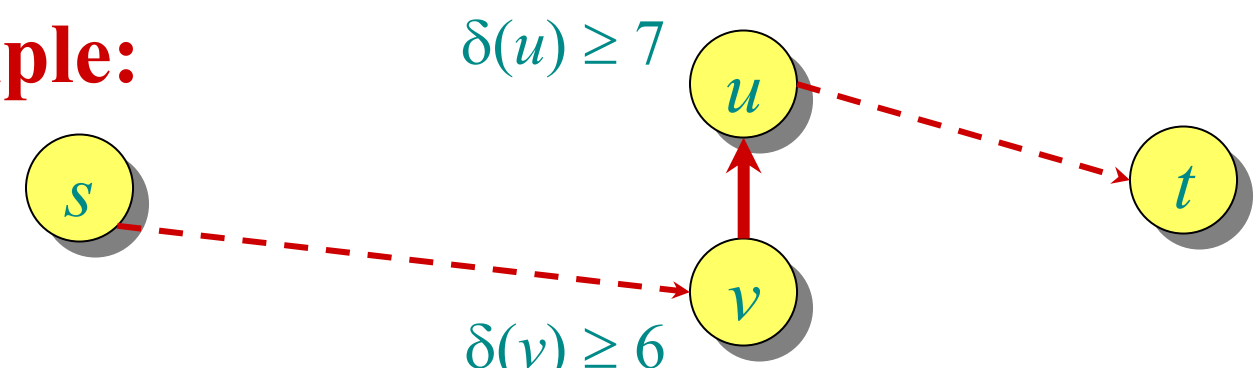


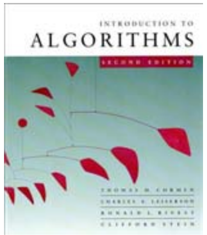
Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned} \delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)} \\ &\geq \delta(v) + 1 && \text{(monotonicity)} \\ &= \delta(u) + 2 && \text{(breadth-first path).} \end{aligned}$$

Example:



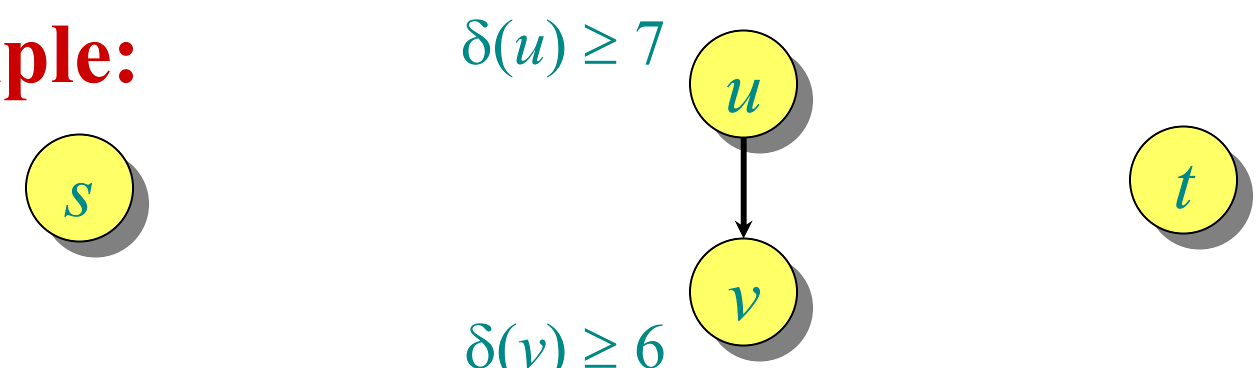


Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned} \delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)} \\ &\geq \delta(v) + 1 && \text{(monotonicity)} \\ &= \delta(u) + 2 && \text{(breadth-first path).} \end{aligned}$$

Example:

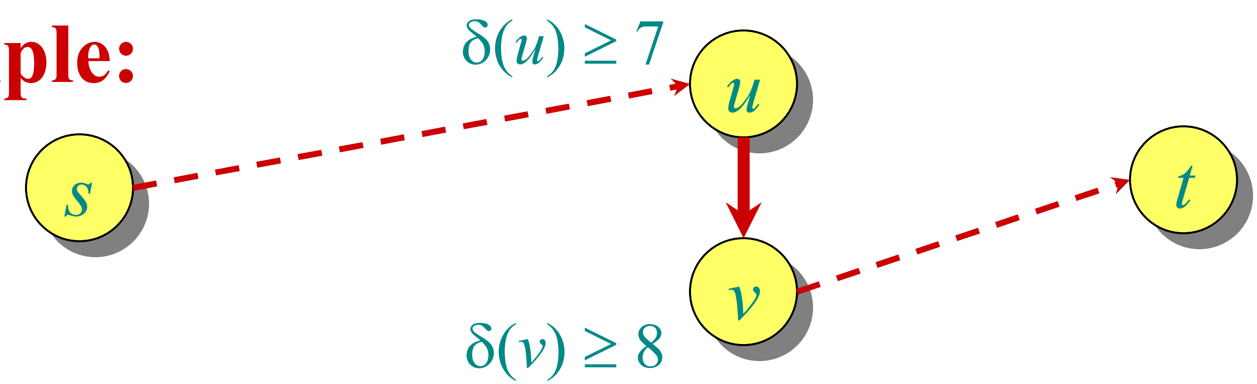


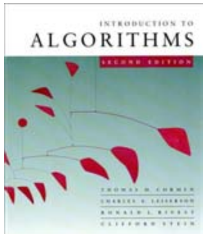
Counting flow augmentations (continued)

The first time an edge (u, v) is critical, we have $\delta(v) = \delta(u) + 1$, since p is a breadth-first path. We must wait until (v, u) is on an augmenting path before (u, v) can be critical again. Let δ' be the distance function when (v, u) is on an augmenting path. Then, we have

$$\begin{aligned} \delta'(u) &= \delta'(v) + 1 && \text{(breadth-first path)} \\ &\geq \delta(v) + 1 && \text{(monotonicity)} \\ &= \delta(u) + 2 && \text{(breadth-first path).} \end{aligned}$$

Example:



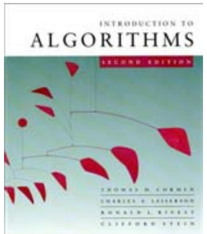


Running time of Edmonds-Karp

Distances start out nonnegative, never decrease, and are at most $|V| - 1$ until the vertex becomes unreachable. Thus, (u, v) occurs as a critical edge $O(V)$ times, because $\delta(v)$ increases by at least 2 between occurrences. Since the residual graph contains $O(E)$ edges, the number of flow augmentations is $O(V E)$. □

Corollary. The Edmonds-Karp maximum-flow algorithm runs in $O(V E^2)$ time.

Proof. Breadth-first search runs in $O(E)$ time, and all other bookkeeping is $O(V)$ per augmentation. □



Best to date

- The asymptotically fastest algorithm to date for maximum flow, due to King, Rao, and Tarjan, runs in $O(|V||E| \log_{|E|/(|V| \log |V|)}|V|)$ time.
- If we allow running times as a function of edge weights, the fastest algorithm for maximum flow, due to Goldberg and Rao, runs in time

$$O\left(\min\{|V|^{2/3}, |E|^{1/2}\} \cdot |E| \log(|V|^2/|E| + 2) \cdot \log C\right),$$

where C is the maximum capacity of any edge in the graph.