Maximum Flow

Contents

Contents.

- **Maximum flow problem.**
- **Minimum cut problem.**
- **Max-flow min-cut theorem.**
- **Augmenting path algorithm.**
- **Capacity-scaling.**
- **Shortest augmenting path.**

Maximum Flow and Minimum Cut

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problem in combinatorial optimization.
- Beautiful mathematical duality.

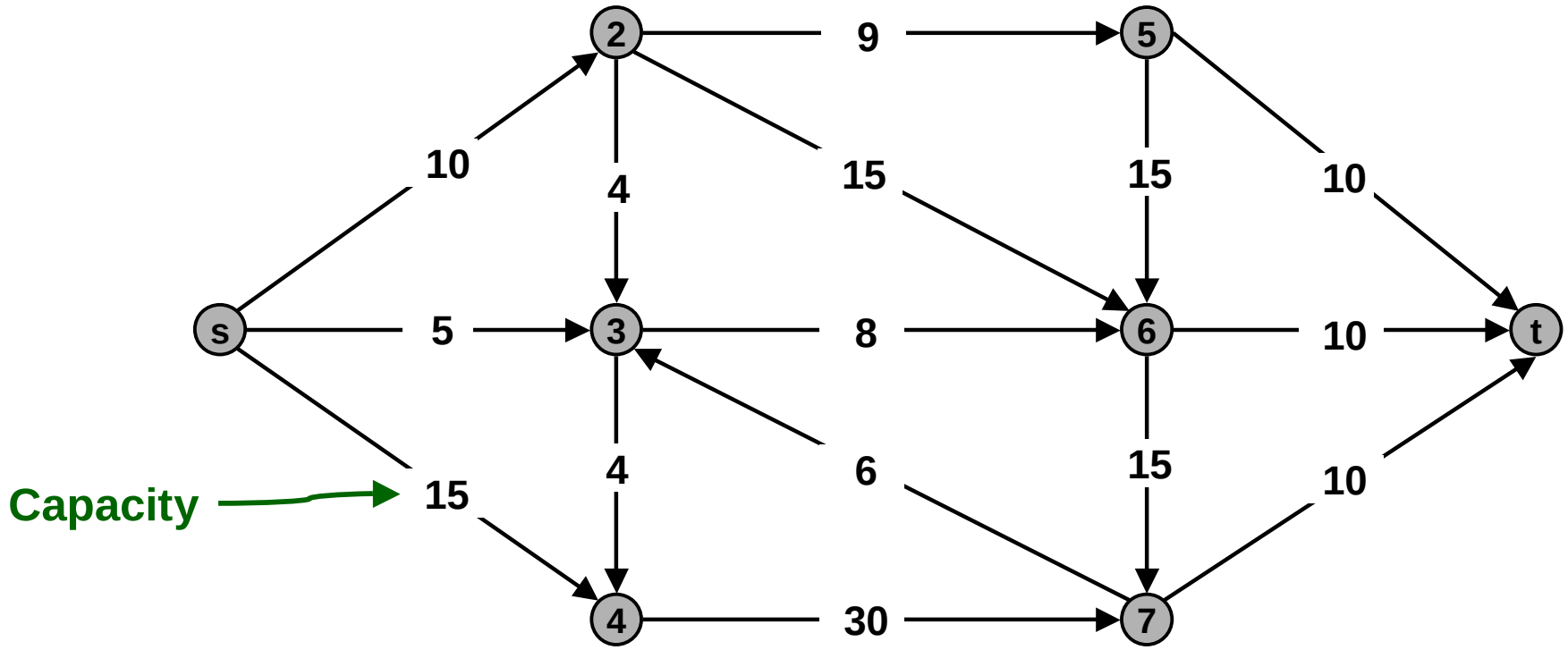
Nontrivial applications / reductions.

- Network connectivity.
- Bipartite matching.
- Data mining.
- Open-pit mining.
- Airline scheduling.
- Image processing.
- Project selection.
- Baseball elimination.
- Network reliability.
- Security of statistical data.
- Distributed computing.
- Egalitarian stable matching.
- Distributed computing.
- Many many more . . .

Max Flow Network

Max flow network: G = (V, E, s, t, u) .

- **(V, E) = directed graph, no parallel arcs.**
- **Two distinguished nodes: s = source, t = sink.**
- **u(e) = capacity of arc e.**



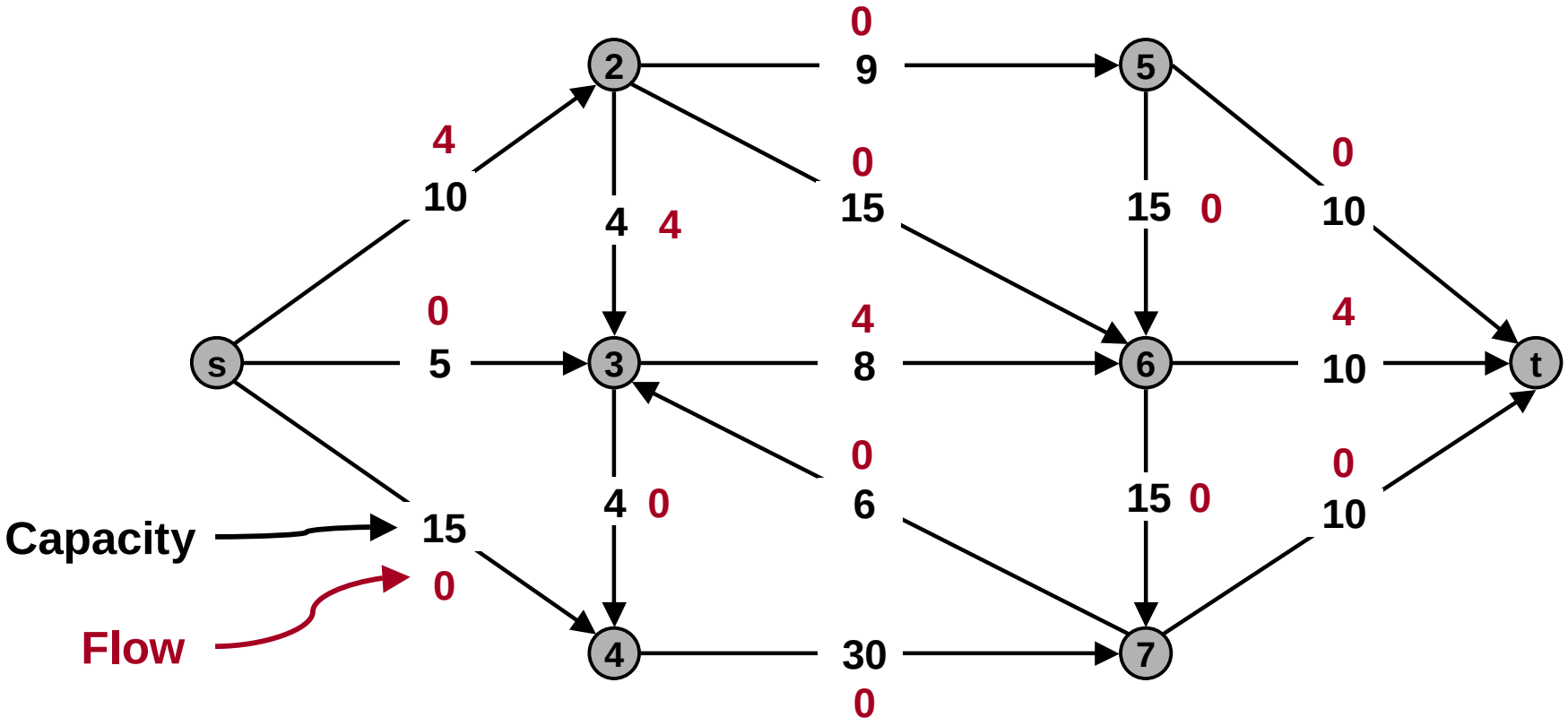
Flows

An s-t flow is a function $f: E \to \Re$ that satisfies:

- For each $e \in E$: $0 \le f(e) \le u(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

$$\sum_{e \text{ in to } v} f(e) := \sum_{w:(w,v) \in E} f(w,v)$$

$$\sum_{e \text{ out of } v} f(e) := \sum_{w:(v,w) \in E} f(v,w)$$



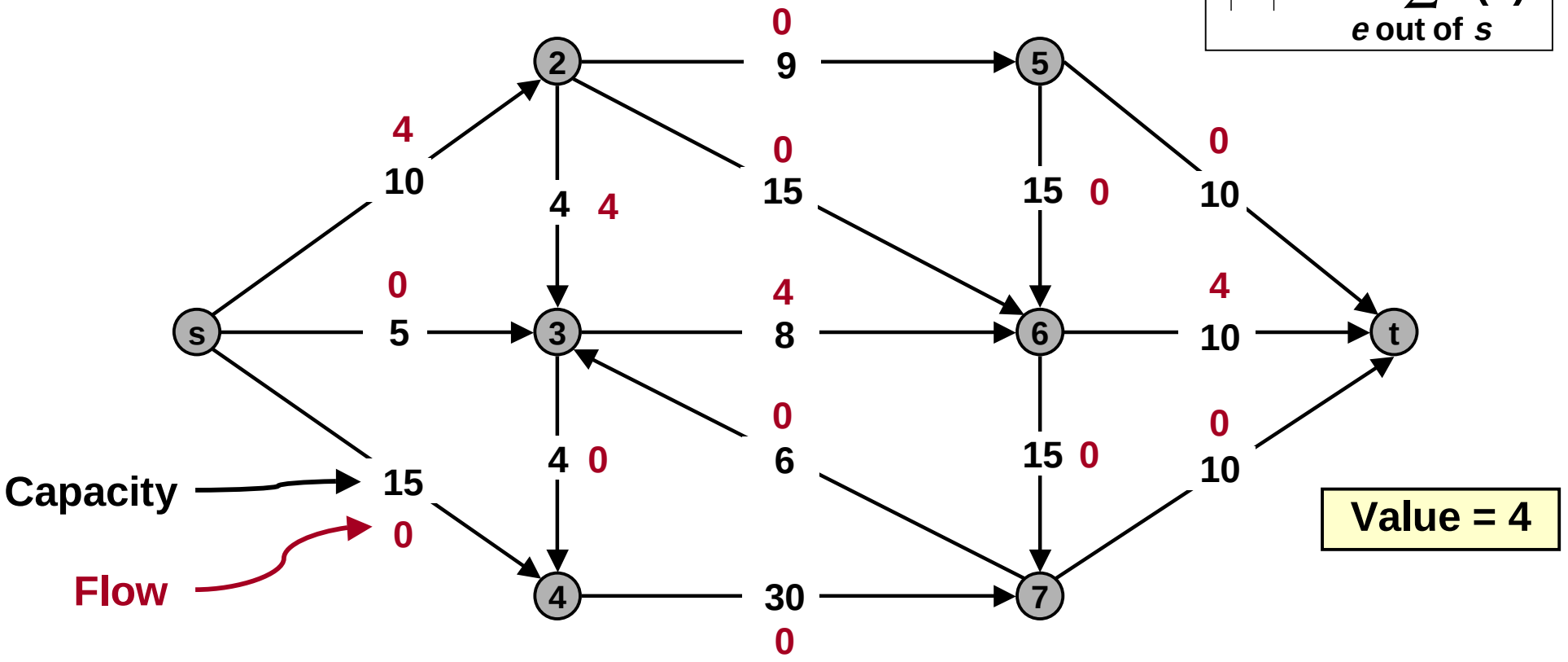
Flows

An s-t flow is a function $f: E \to \Re$ that satisfies:

- **For each $e \in E$:** $0 \le f(e) \le u(e)$ **(capacity)**
- **For each $v \in V - \{s, t\}$:** $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ **(conservation)**

MAX FLOW: find s-t flow that maximizes net flow out of the source.

$$|f| = \sum_{e \text{ out of } s} f(e)$$

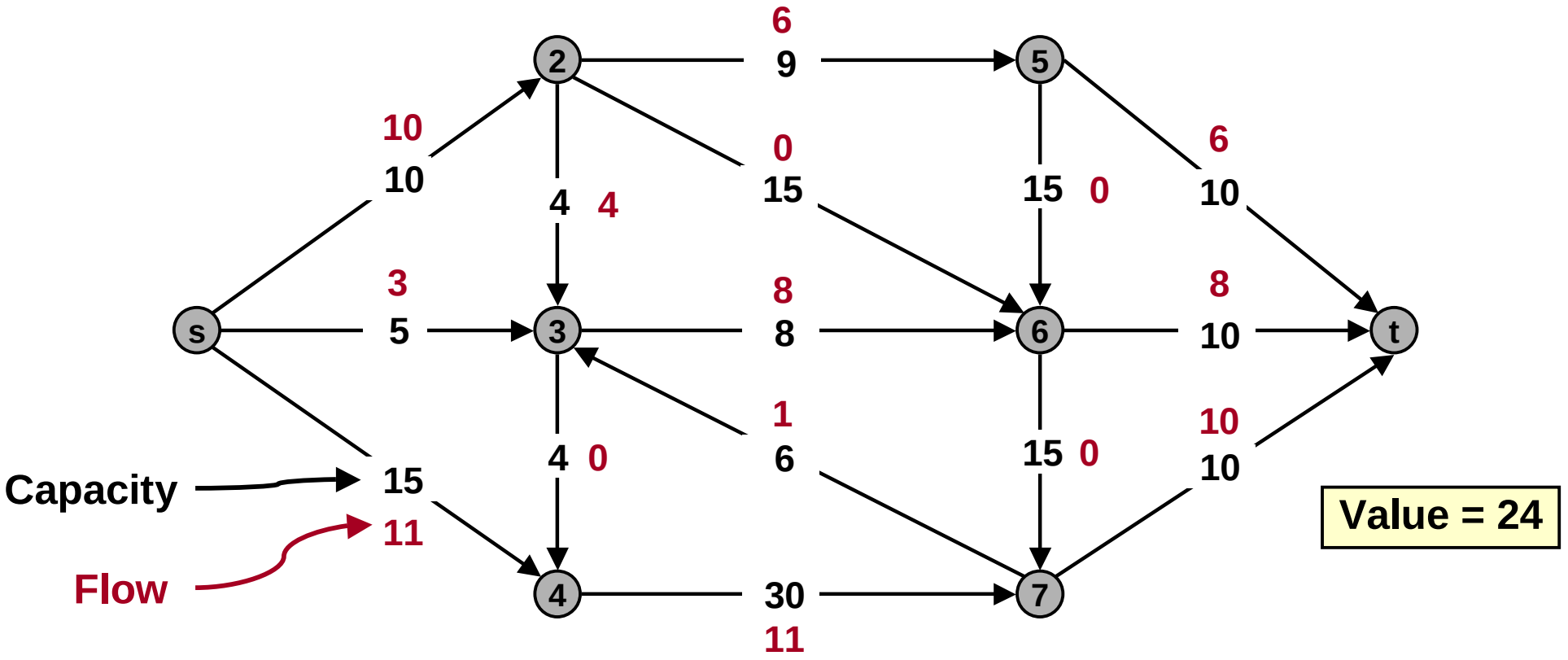


Flows

An s-t flow is a function $f: E \to \Re$ that satisfies:

- **For each $e \in E$:** $0 \le f(e) \le u(e)$ **(capacity)**
- **For each $v \in V - \{s, t\}$:** $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ **(conservation)**

MAX FLOW: find s-t flow that maximizes net flow out of the source.

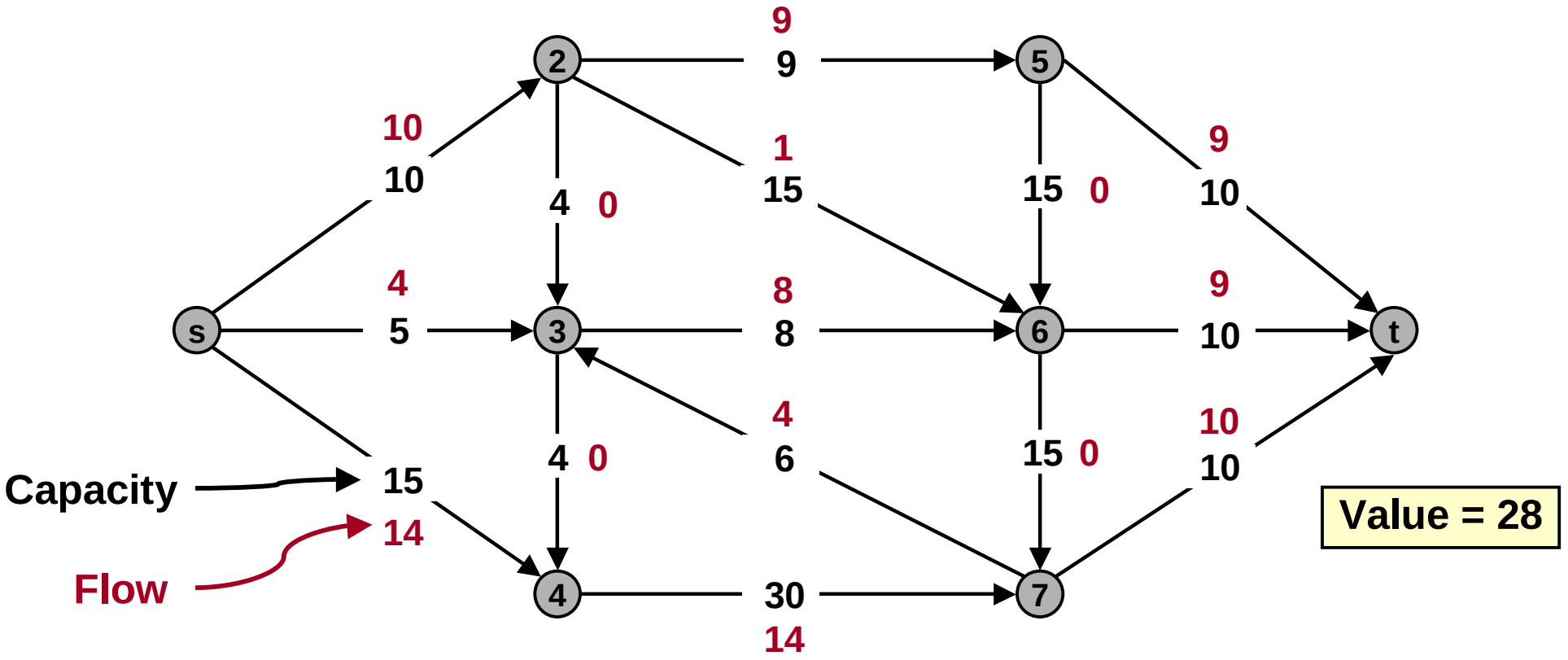


Flows

An s-t flow is a function $f: E \to \Re$ that satisfies:

- **For each $e \in E$:** $0 \le f(e) \le u(e)$ **(capacity)**
- **For each $v \in V - \{s, t\}$:** $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ **(conservation)**

MAX FLOW: find s-t flow that maximizes net flow out of the source.



Networks

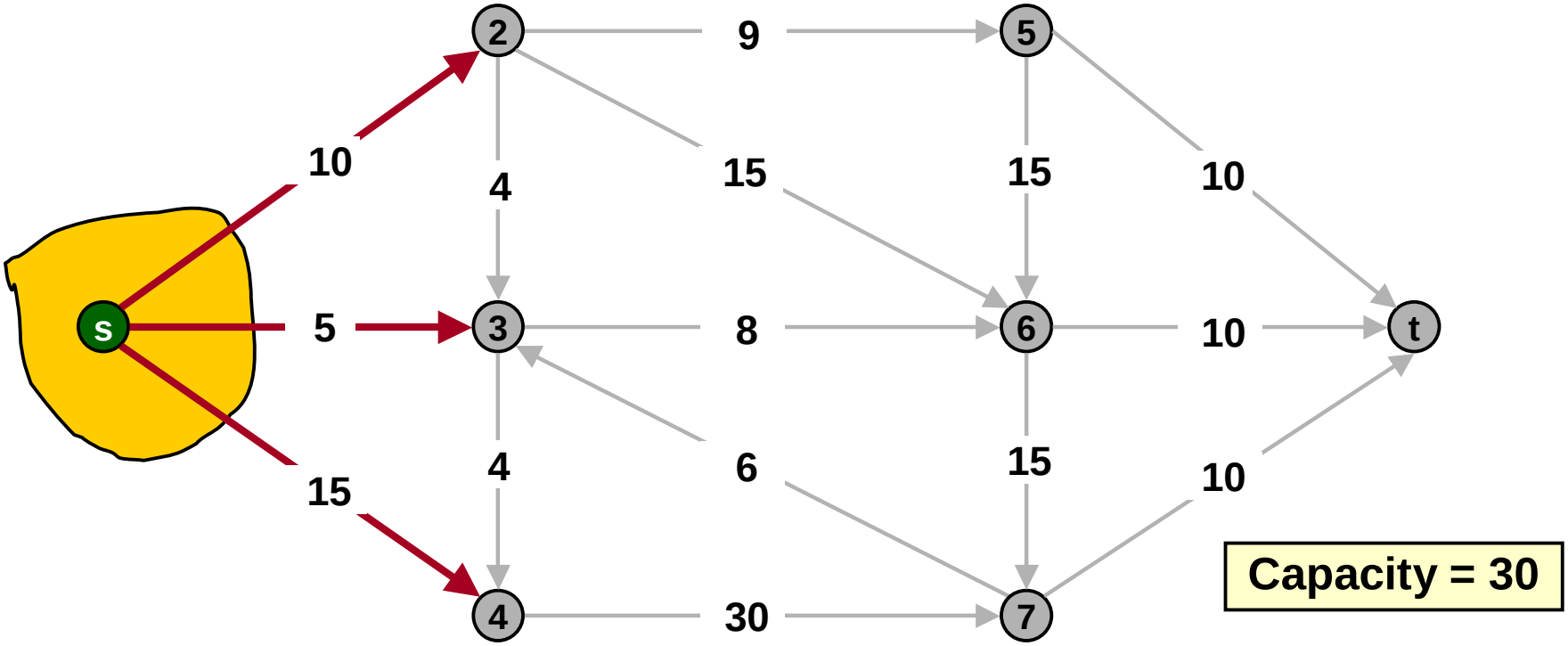
Network	Nodes	Arcs	Flow
communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
circuits	gates, registers, processors	wires	current
mechanical	joints	rods, beams, springs	heat, energy
hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
financial	stocks, currency	transactions	money
transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers
chemical	sites	bonds	energy

Cuts

An s-t cut is a node partition (S, T) such that $s \in S$, $t \in T$.

- **The capacity of an s-t cut (S, T) is:** $\sum_{e \text{ out of } S} u(e) := \sum_{\substack{(v,w) \in E \\ v \in S,\ w \in T}} u(v,w).$

Min s-t cut: find an s-t cut of minimum capacity.

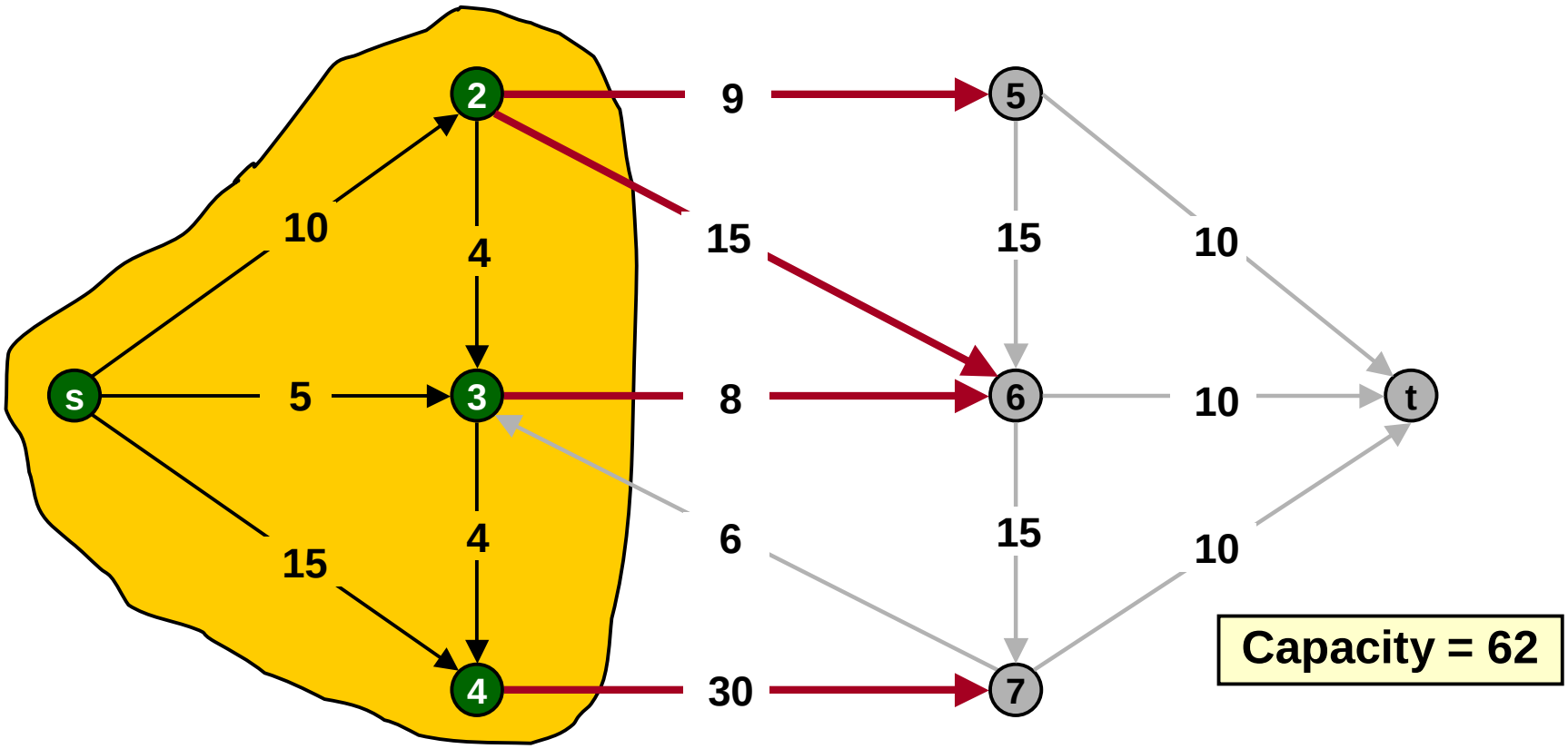


Cuts

An s-t cut is a node partition (S, T) such that s ∈ S, t ∈ T.

- **The capacity of an s-t cut (S, T) is:** $\sum_{e \text{ out of } S} u(e) := \sum_{\substack{(v,w) \in E \\ v \in S,\ w \in T}} u(v,w).$

Min s-t cut: find an s-t cut of minimum capacity.

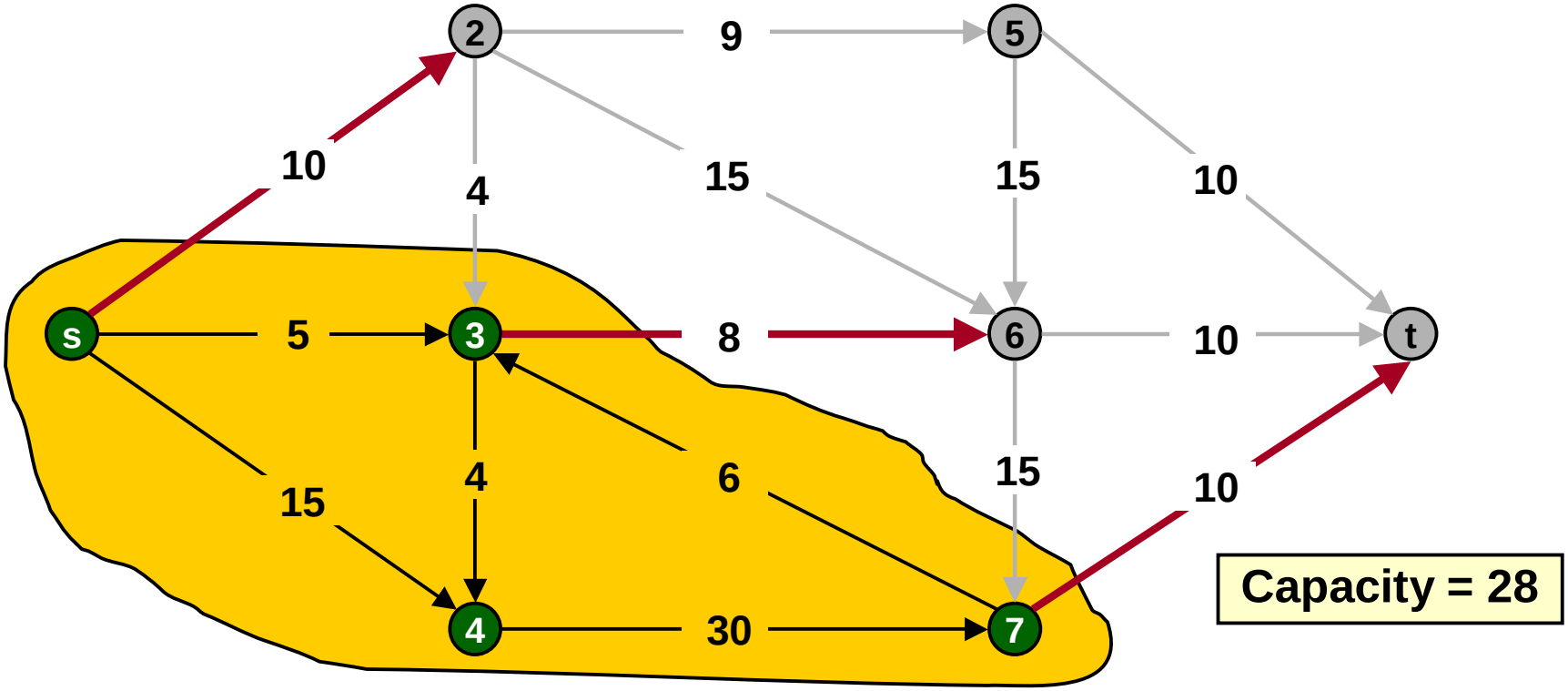


Cuts

An s-t cut is a node partition (S, T) such that s ∈ S, t ∈ T.

- **The capacity of an s-t cut (S, T) is:** $\sum_{e \text{ out of } S} u(e) := \sum_{\substack{(v,w) \in E \\ v \in S,\ w \in T}} u(v,w).$

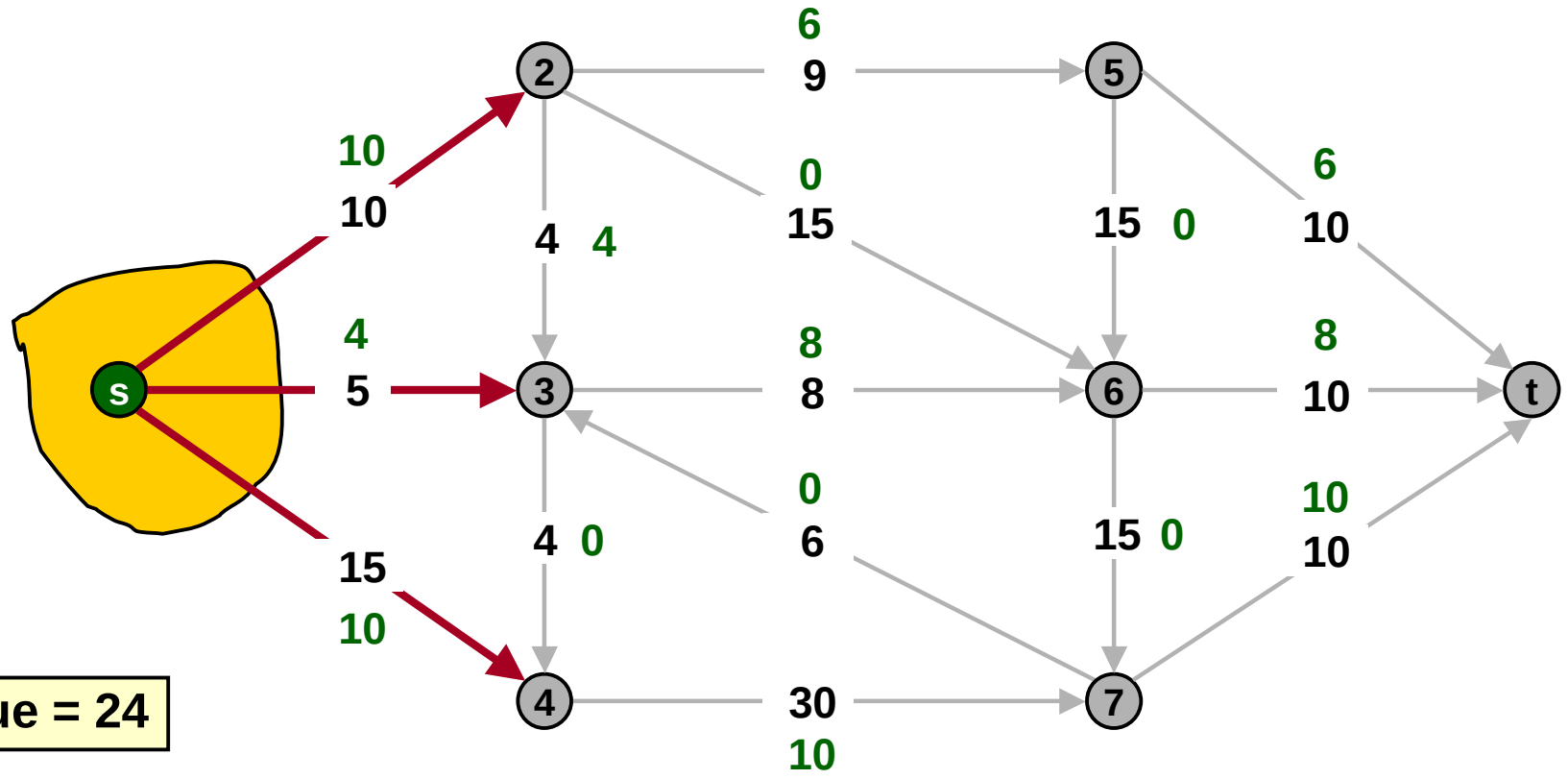
Min s-t cut: find an s-t cut of minimum capacity.



Flows and Cuts

L1. Let f be a flow, and let (S, T) be a cut. Then, the net flow sent across the cut is equal to the amount reaching t.

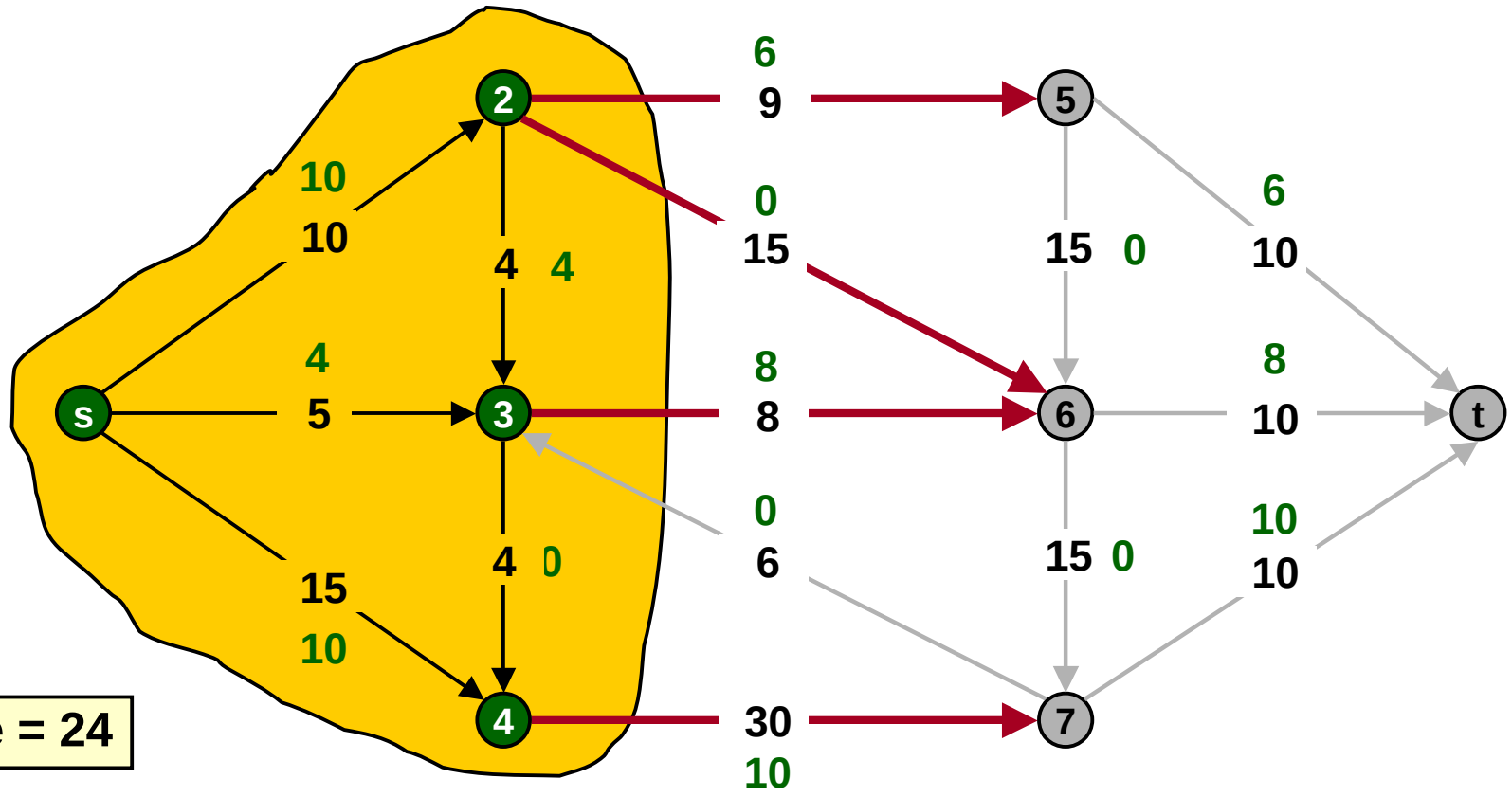
$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$



Flows and Cuts

L1. Let f be a flow, and let (S, T) be a cut. Then, the net flow sent across the cut is equal to the amount reaching t.

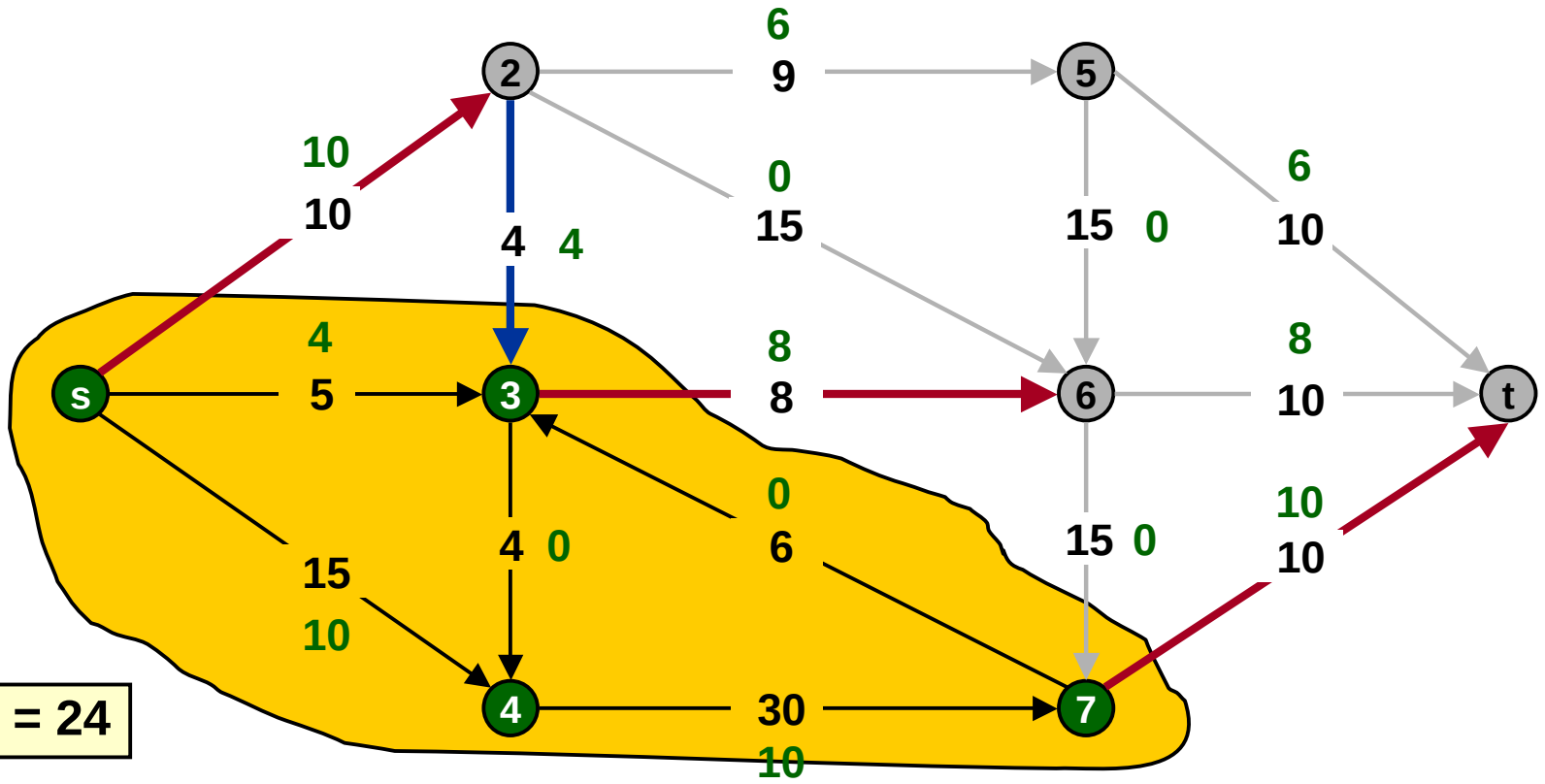
$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$



Flows and Cuts

L1. Let f be a flow, and let (S, T) be a cut. Then, the net flow sent across the cut is equal to the amount reaching t.

$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$



Value = 24

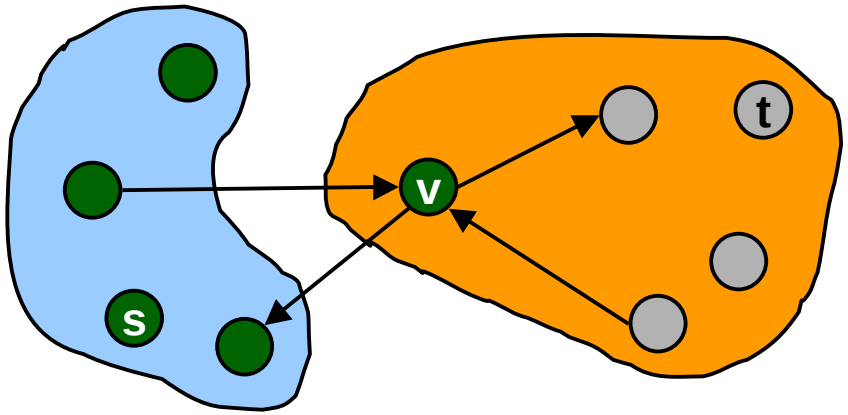
Flows and Cuts

Let f be a flow, and let (S, T) be a cut. Then, $\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = |f|.$

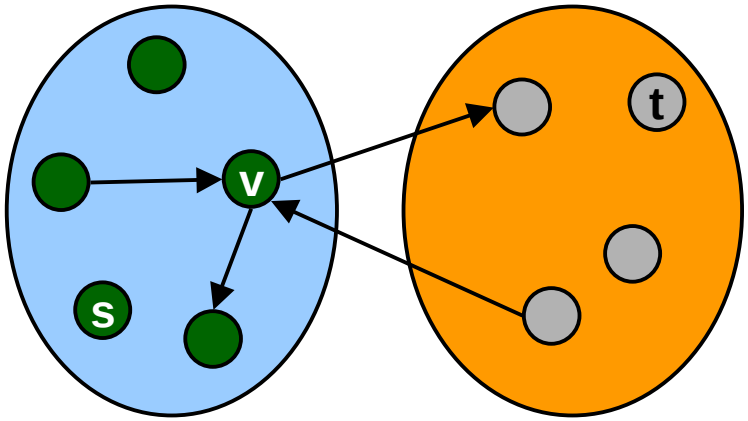
Proof by induction on |S|.

- **Base case: S = { s }.**
- **Inductive hypothesis: assume true for |S| < k.**
  - **consider cut (S, T) with |S| = k**
  - **S = S' $\cup$ { v } for some v $\neq$ s, t, |S' | = k-1 $\Rightarrow$ cap(S', T') = | f |.**
  - **adding v to S' increase cut capacity by**

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) = 0$$



S' Before



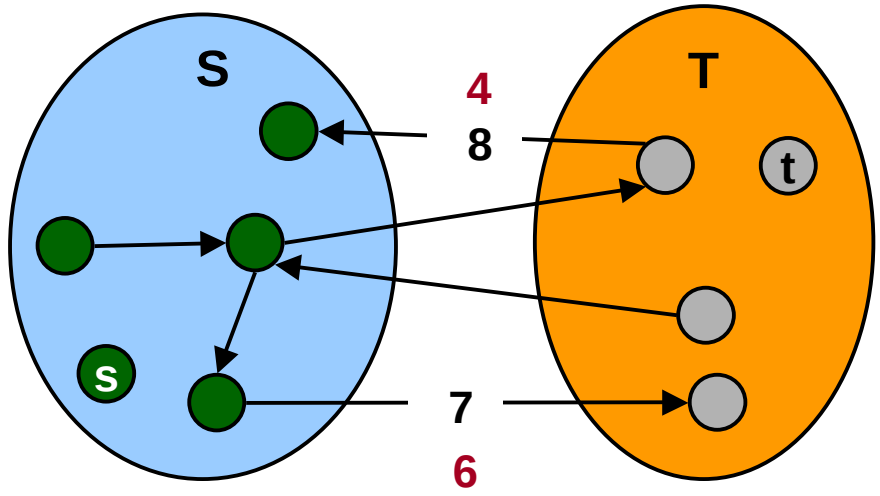
S After

Flows and Cuts

L2. Let f be a flow, and let (S, T) be a cut. Then, $| f | \leq \text{cap}(S, T)$.

Proof.

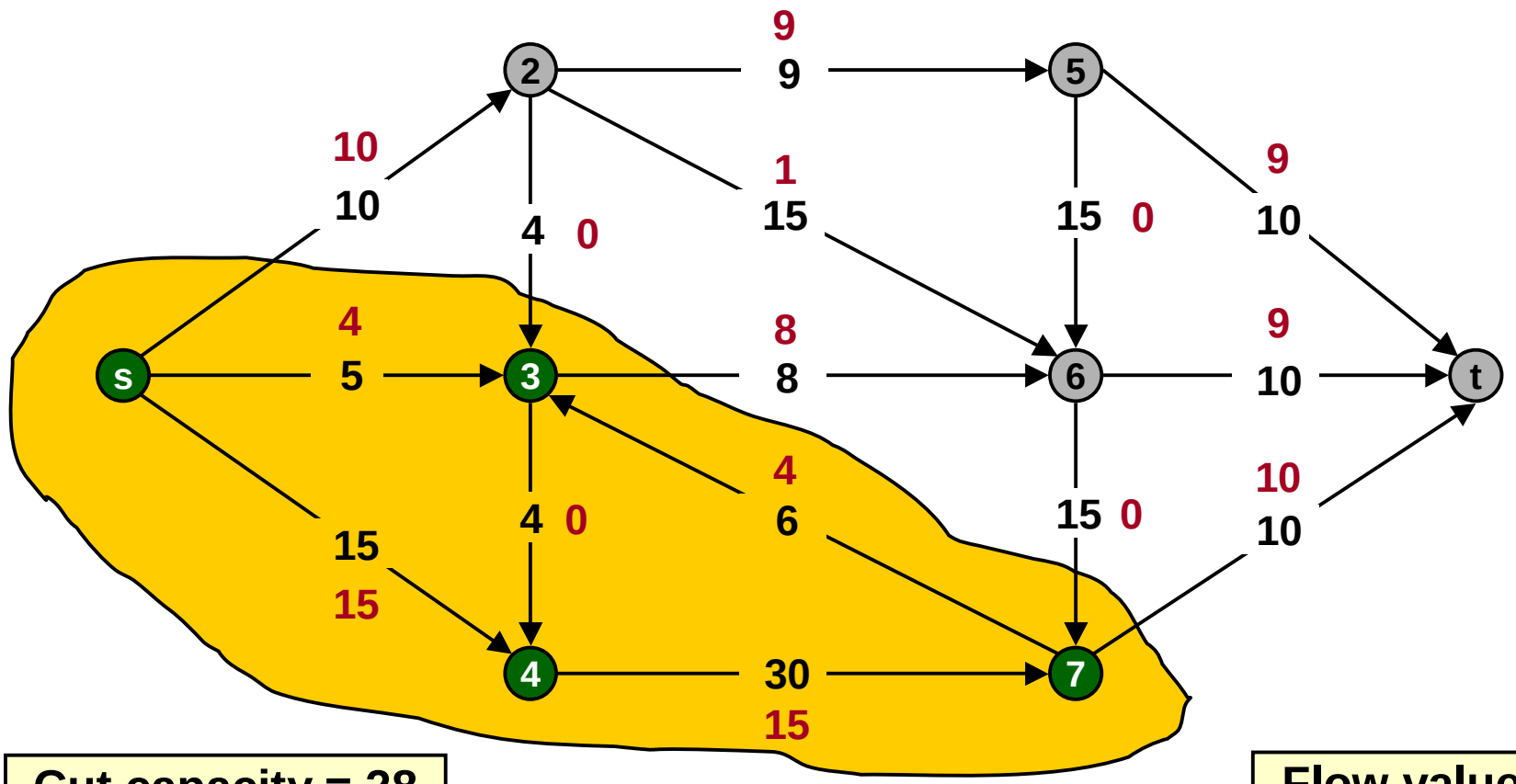
$$\begin{aligned} |f| &= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\ &\leq \sum_{e \text{ out of } S} f(e) \\ &\leq \sum_{e \text{ out of } S} u(e) \\ &= \text{cap}(S, T) \end{aligned}$$



Corollary. Let f be a flow, and let (S, T) be a cut. If |f| = cap(S, T), then f is a max flow and (S, T) is a min cut.

Max Flow and Min Cut

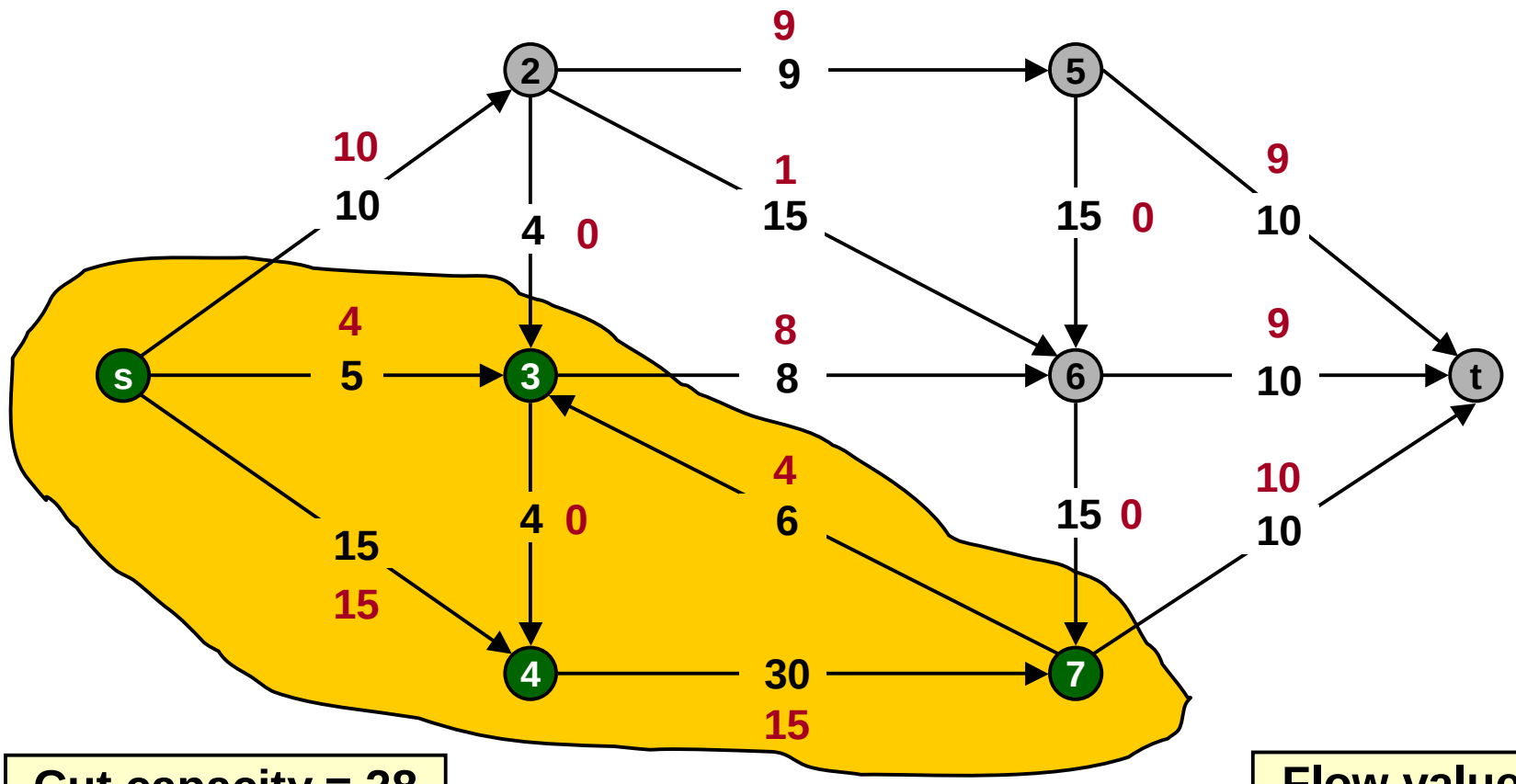
Corollary. Let f be a flow, and let (S, T) be a cut. If |f| = cap(S, T), then f is a max flow and (S, T) is a min cut.



Max-Flow Min-Cut Theorem

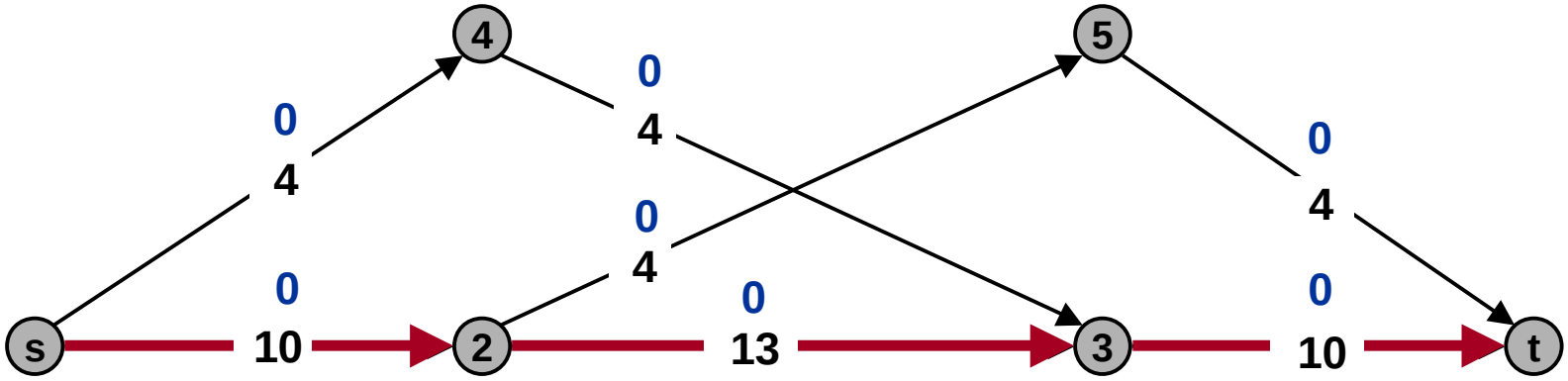
MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): In any network, the value of the max flow is equal to the value of the min cut.

- "Good characterization."
- Proof IOU.



Towards an Algorithm

Find an s-t path where each arc has u(e) > f(e) and "augment" flow along the path.

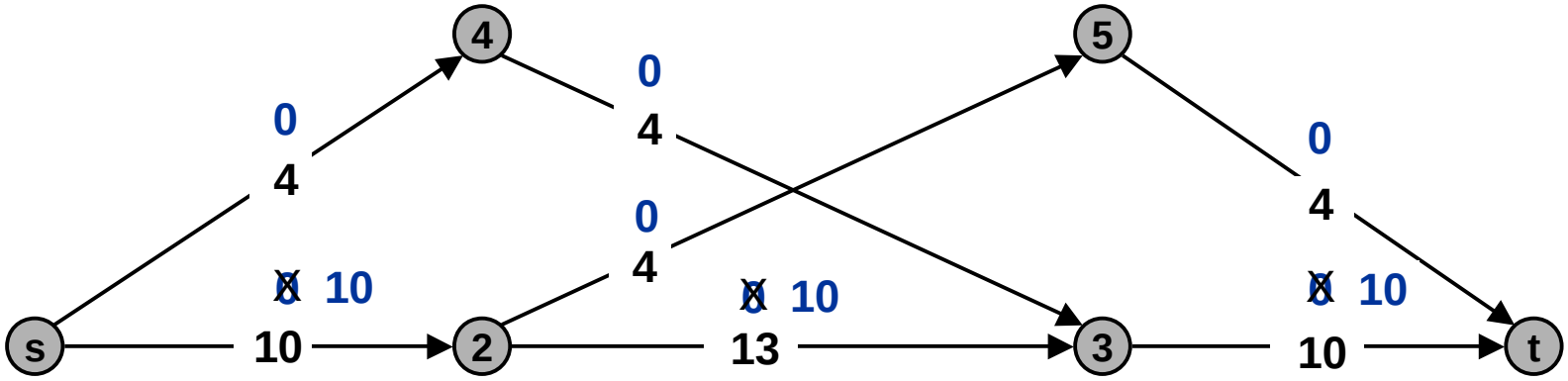


Flow value = 0

Towards an Algorithm

Find an s-t path where each arc has u(e) > f(e) and "augment" flow along the path.

- **Repeat until you get stuck.**

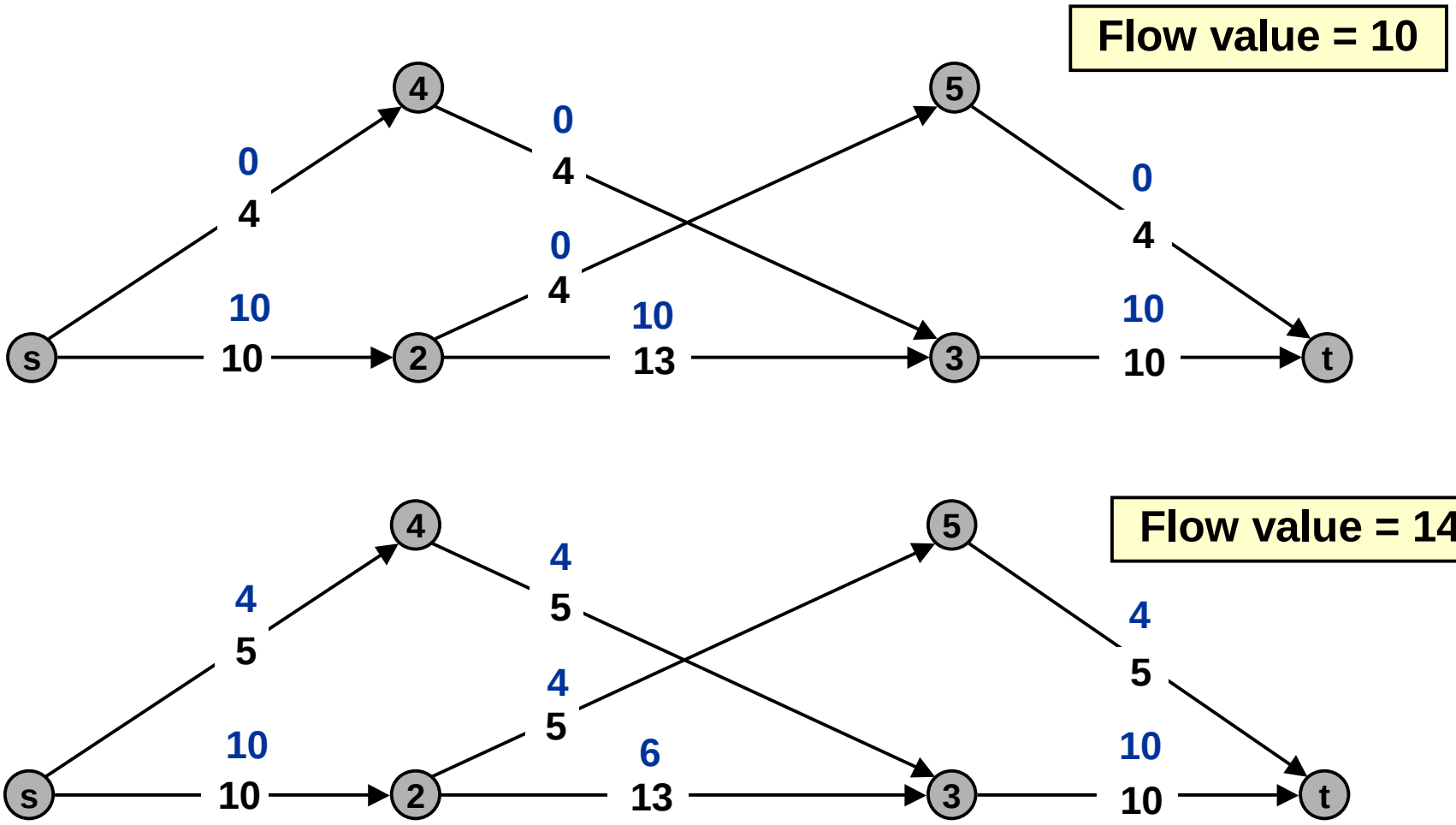


Flow value = 10

Towards an Algorithm

Find an s-t path where each arc has u(e) > f(e) and "augment" flow along the path.

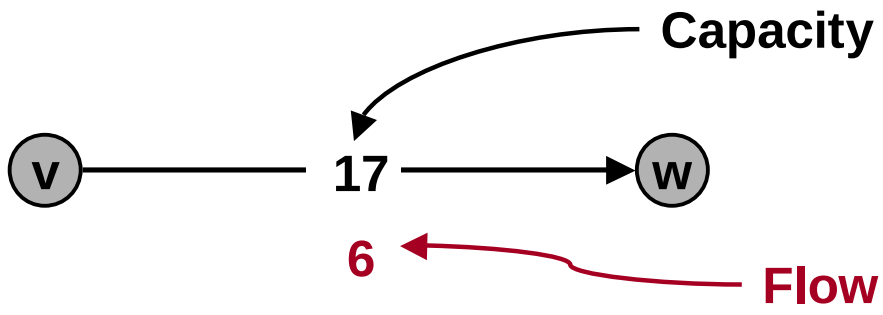
- **Repeat until you get stuck.**
- **Greedy algorithm fails.**



Residual Arcs

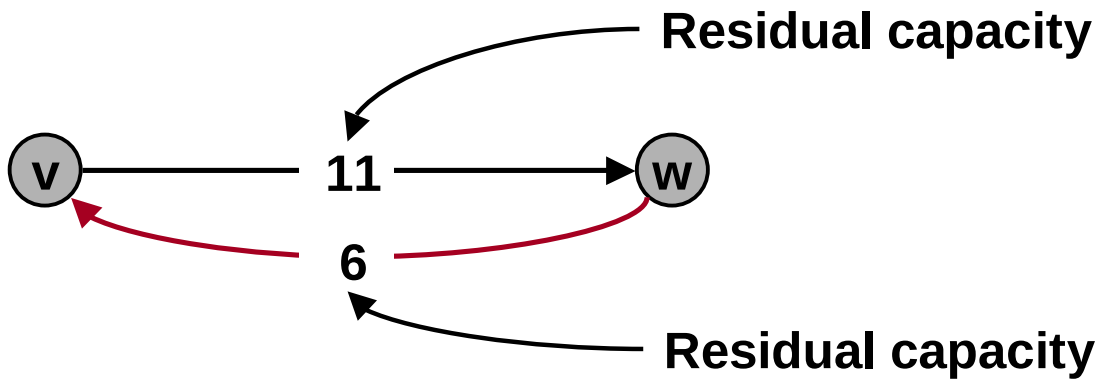
Original graph G = (V, E).

- **Flow f(e).**
- **Arc e = (v, w) $\in$ E.**



Residual graph: $G_f = (V, E_f)$.

- **Residual arcs e = (v, w) and e^R = (w, v).**
- **"Undo" flow sent.**

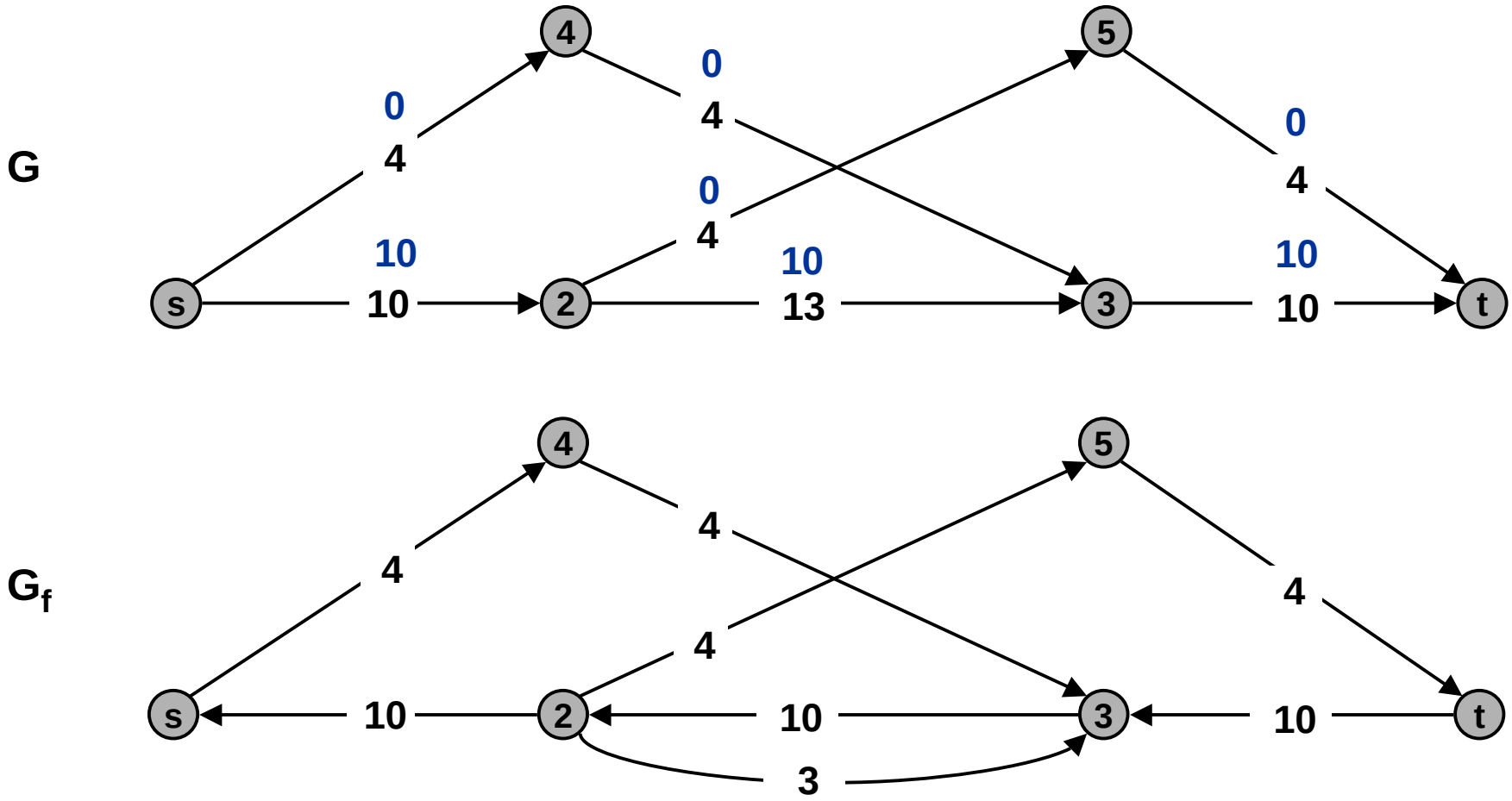


Residual Graph and Augmenting Paths

Residual graph: $G_f = (V, E_f)$.

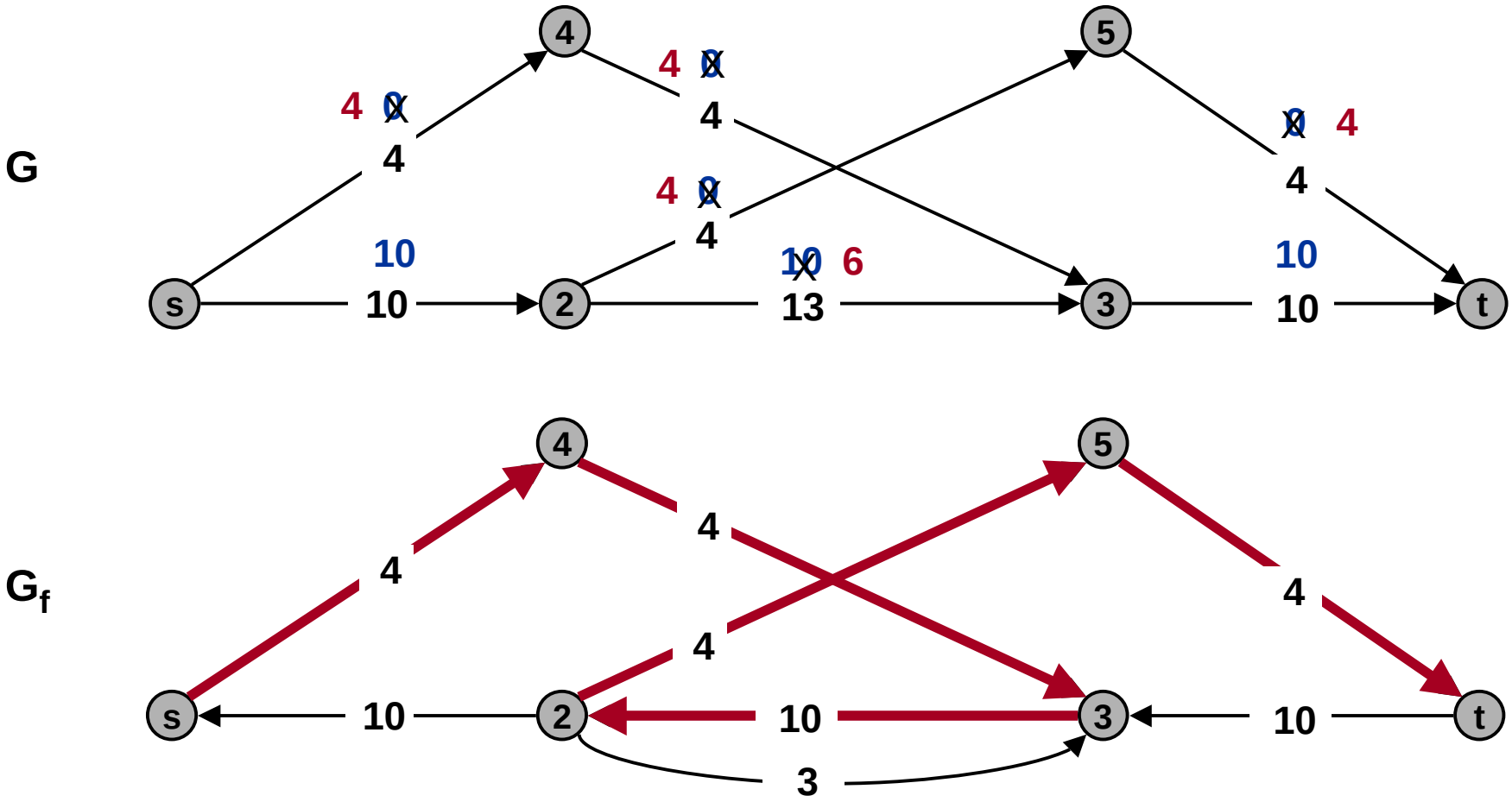
- $E_f = \{e : f(e) < u(e)\} \cup \{e^R : f(e) > 0\}$.

$$u_f(e) = \begin{cases} u(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



Augmenting Path

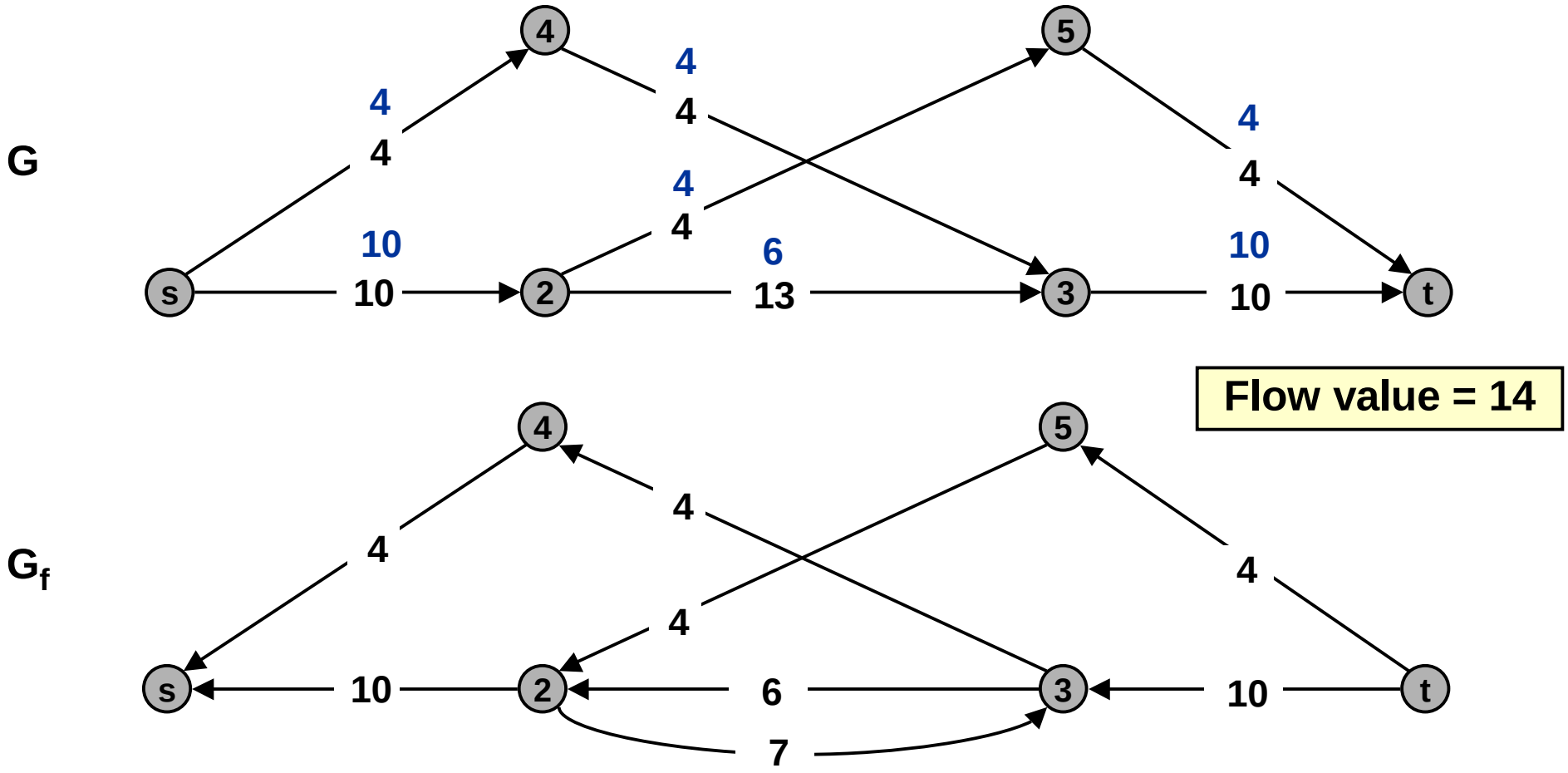
Augmenting path = path in residual graph.



Augmenting Path

Augmenting path = path in residual graph.

- **Max flow ⇔ no augmenting paths ???**



Max-Flow Min-Cut Theorem

Augmenting path theorem (Ford-Fulkerson, 1956): A flow f is a max flow if and only if there are no augmenting paths.

MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): the value of the max flow is equal to the value of the min cut.

We prove both simultaneously by showing the TFAE:

(i) **f is a max flow.**

(ii) **There is no augmenting path relative to f.**

(iii) **There exists a cut (S, T) such that |f| = cap(S, T).**

Proof of Max-Flow Min-Cut Theorem

We prove both simultaneously by showing the TFAE:

(i) **f is a max flow.**

(ii) **There is no augmenting path relative to f.**

(iii) **There exists a cut (S, T) such that |f| = cap(S, T).**

(i) $\Rightarrow$ (ii)

- **We show contrapositive.**
- **Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.**

(iii) $\Rightarrow$ (i)

- **Next slide.**

(iii) $\Rightarrow$ (i)

- **This was the Corollary to Lemma 2.**

Proof of Max-Flow Min-Cut Theorem

We prove both simultaneously by showing the TFAE:

(i) f is a max flow.

(ii) There is no augmenting path relative to f.

(iii) There exists a cut (S, T) such that |f| = cap(S, T).

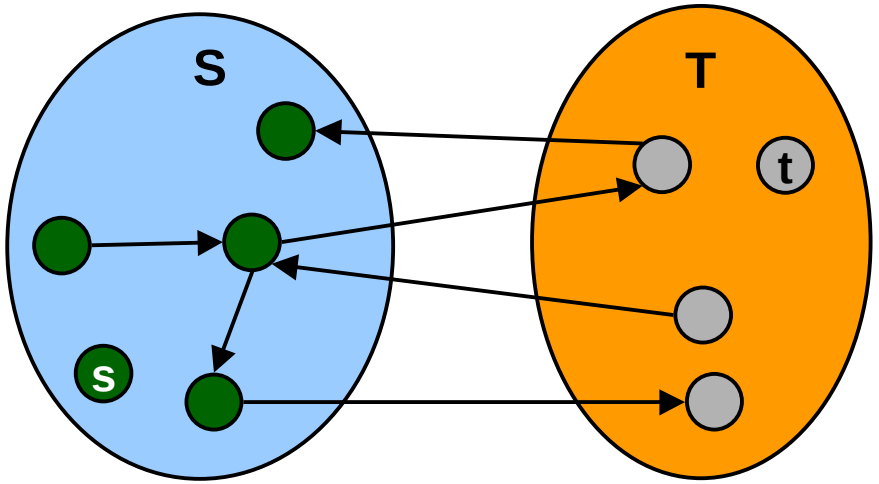
(ii) $\Rightarrow$ (iii)

- Let f be a flow with no augmenting paths.
- Let S be set of vertices reachable from s in residual graph.
  - clearly s $\in$ S, and t $\notin$ S by definition of f

$$|f| = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e)$$

$$= \sum_{e \text{ out of } S} u(e)$$

$$= cap(S, T)$$

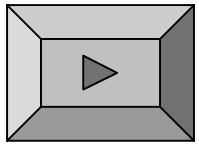


Original Network

Augmenting Path Algorithm

Augment (f, P)

```
b ← bottleneck(P)
FOREACH e ∈ P
   IF (e ∈ E) // forward arc
      f(e) ← f(e) + b
   ELSE       // backwards arc
      f(e^R) ← f(e) - b
RETURN f
```



FordFulkerson (V, E, s, t)

```
FOREACH e ∈ E
   f(e) ← 0
G_f ← residual graph

WHILE (there exists augmenting path P)
   f ← augment(f, P)
   update G_f
RETURN f
```

Running Time

Assumption: all capacities are integers between 0 and U.

Invariant: every flow value f(e) and every residual capacities $u_f(e)$ remains an integer throughout the algorithm.

Theorem: the algorithm terminates in at most $|f^*| \leq nU$ iterations.

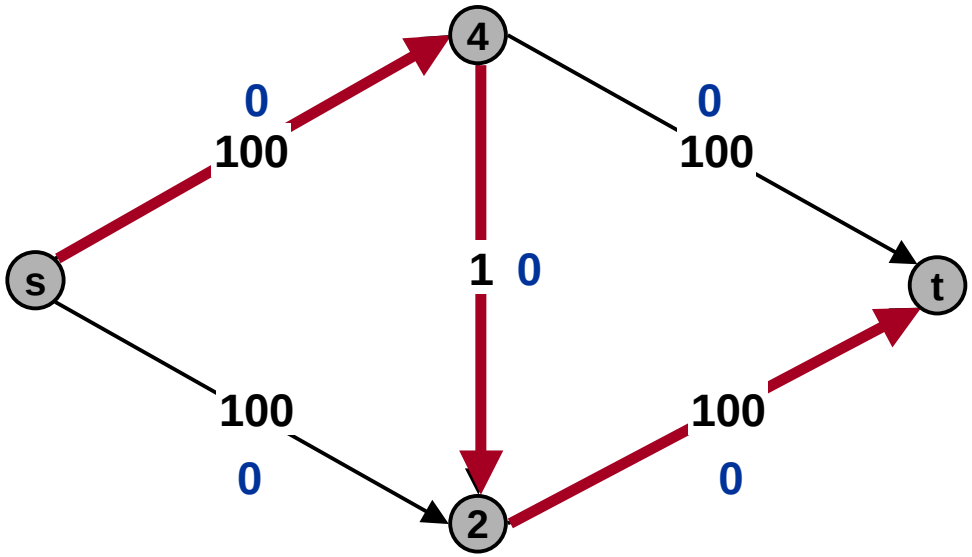
Corollary: if U = 1, then algorithm runs in O(mn) time.

Integrality theorem: if all arc capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.

Note: algorithm may not terminate on some pathological instances (with irrational capacities). Moreover, flow value may not even converge to correct answer.

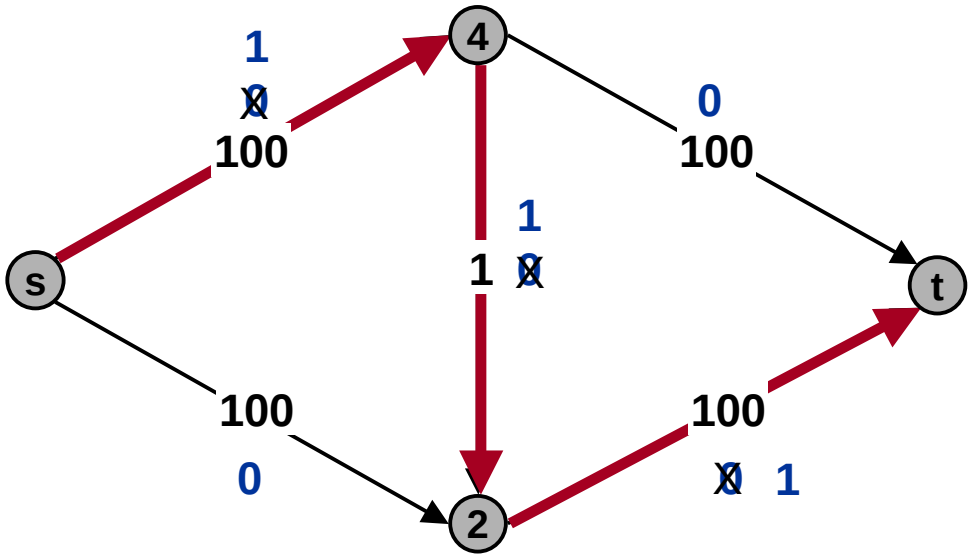
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



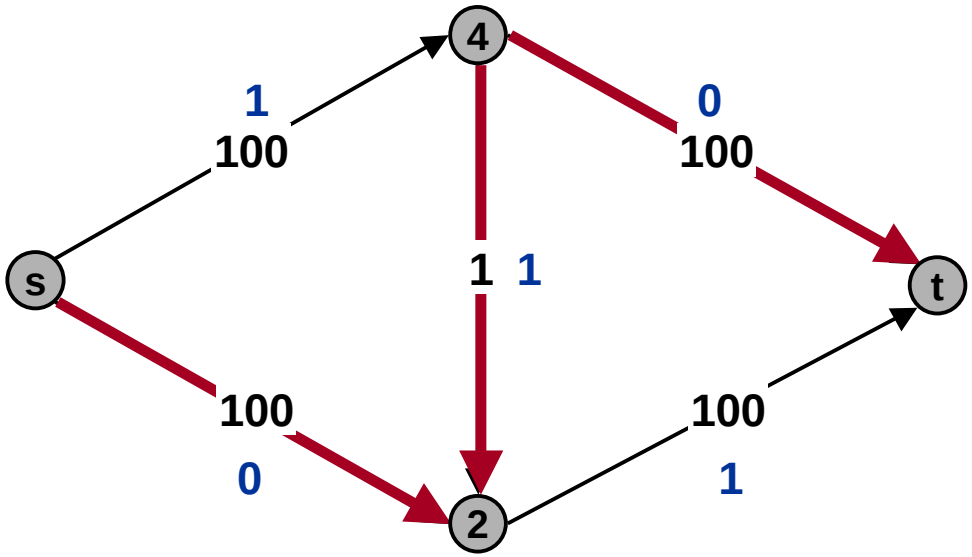
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



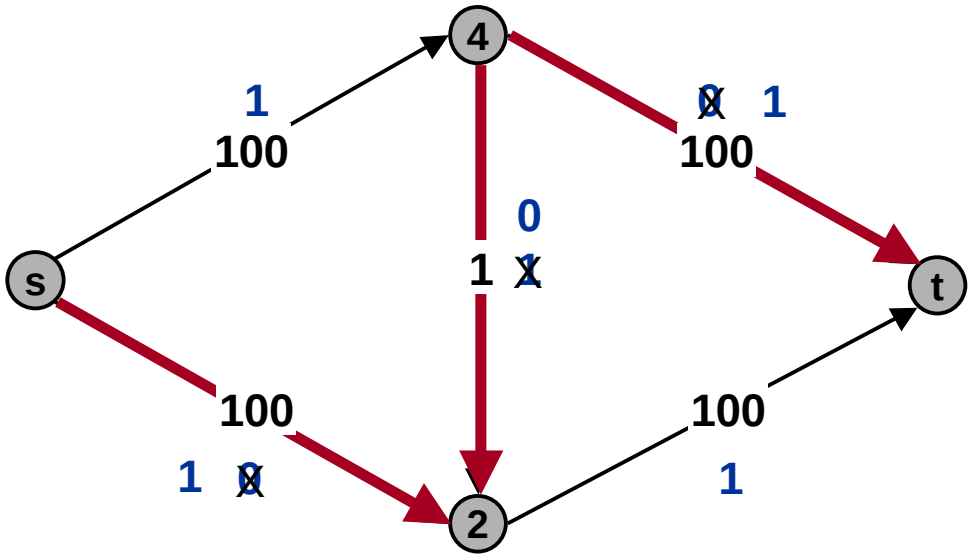
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



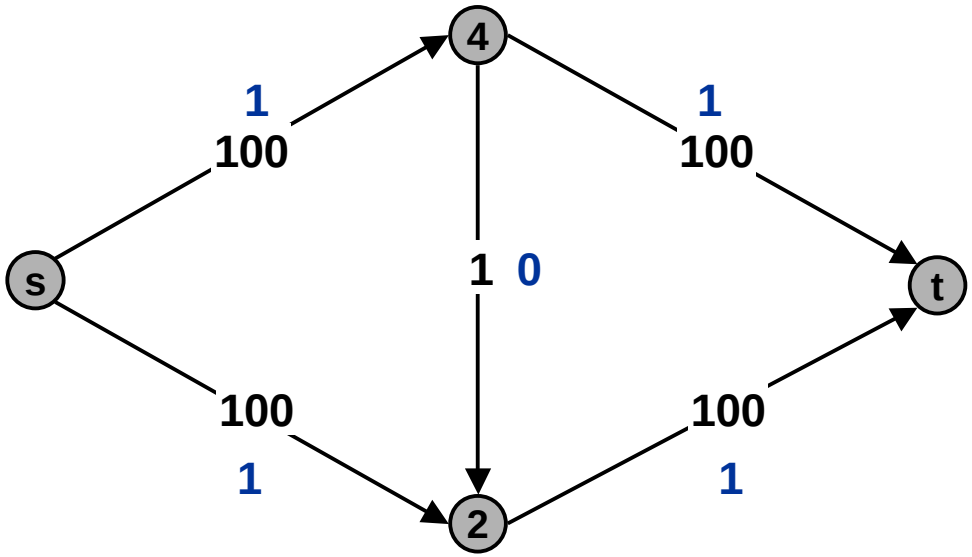
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



200 iterations possible.

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- **Some choices lead to exponential algorithms.**
- **Clever choices lead to polynomial algorithms.**

Goal: choose augmenting paths so that:

- **Can find augmenting paths efficiently.**
- **Few iterations.**

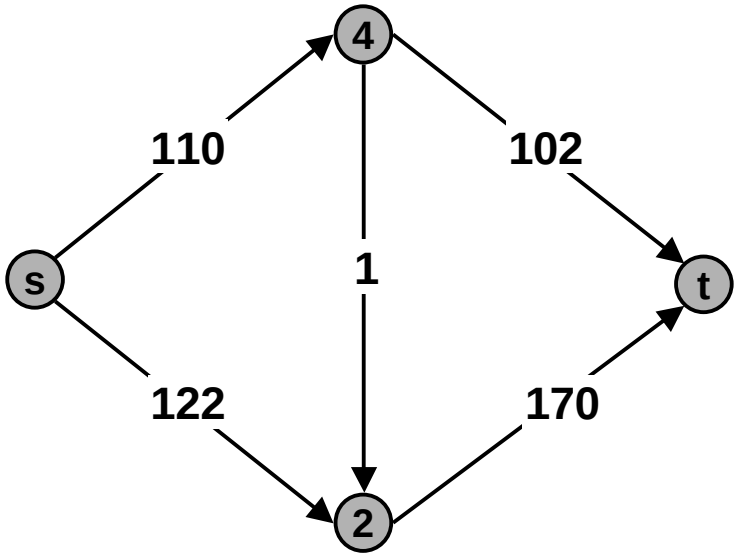
Edmonds-Karp (1972): choose augmenting path with

- **Max bottleneck capacity.** **(fat path)**
- **Sufficiently large capacity.** **(capacity-scaling)**
- **Fewest number of arcs.** **(shortest path)**

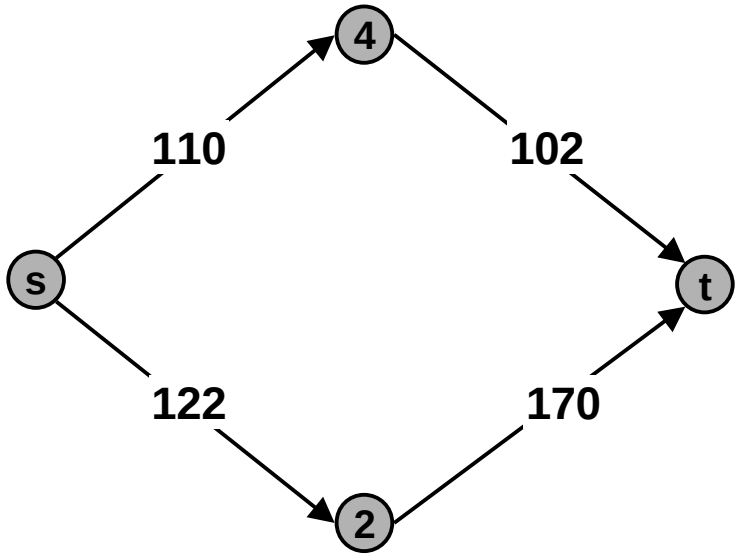
Capacity Scaling

Intuition: choosing path with highest bottleneck capacity increases flow by max possible amount.

- **Don't worry about finding exact highest bottleneck path.**
- **Maintain scaling parameter Δ.**
- **Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.**



G_f



$G_f(100)$

Capacity Scaling

Intuition: choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.

ScalingMaxFlow(V, E, s, t)

```
FOREACH e ∈ E, f(e) ← 0
Δ ← smallest power of 2 greater than or equal to U

WHILE (Δ ≥ 1)
   Gf(Δ) ← Δ-residual graph
   WHILE (there exists augmenting path P in Gf(Δ))
      f ← augment(f, P)
      update Gf(Δ)
   Δ ← Δ / 2
RETURN f
```

Capacity Scaling: Analysis

L1. If all arc capacities are integers, then throughout the algorithm, all flow and residual capacity values remain integers.

- Thus, $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$, so upon termination f is a max flow.

L2. The outer while loop repeats $1 + \lfloor \log_2 U \rfloor$ times.

- Initially $U \leq \Delta < 2U$, and Δ decreases by a factor of 2 each iteration.

L3. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $| f | + m \Delta$.

L4. There are at most 2m augmentations per scaling phase.

- Let f be the flow at the end of the previous scaling phase.
- L3 $\Rightarrow$ $|f^*| \leq | f | + m (2\Delta)$.
- Each augmentation in a Δ-phase increases $| f |$ by at least Δ.

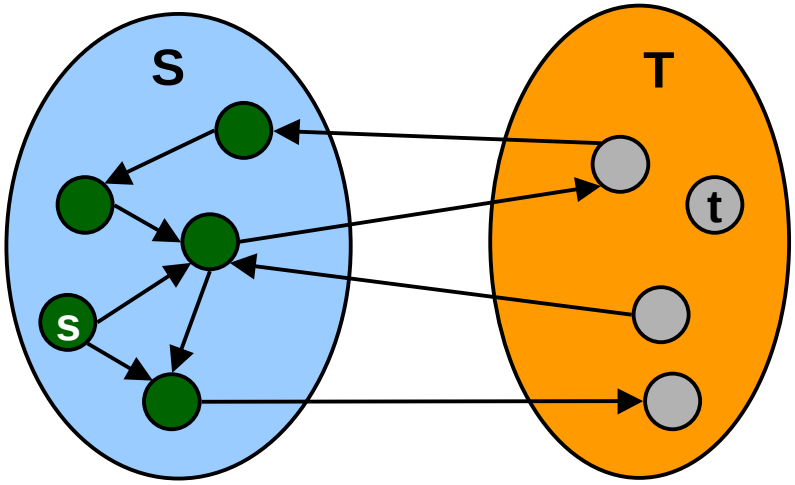
Theorem. The algorithm runs in $O(m^2 \log (2U))$ time.

Capacity Scaling: Analysis

L3. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $|f| + m\,\Delta$.

- **We show that at the end of a Δ-phase, there exists a cut (S, T) such that $\text{cap}(S, T) \leq |f| + m\,\Delta$.**
- **Choose S to be the set of nodes reachable from s in $G_f(\Delta)$.**
  - **clearly $s \in S$, and $t \notin S$ by definition of S**

$$
\begin{aligned}
|f| &= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\
&\geq \sum_{e \text{ out of } S} (u(e) - \Delta) - \sum_{e \text{ in to } S} \Delta \\
&= \sum_{e \text{ out of } S} u(e) - \sum_{e \text{ out of } S} \Delta - \sum_{e \text{ in to } S} \Delta \\
&= cap(S, T) - m\Delta
\end{aligned}
$$



Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- **Some choices lead to exponential algorithms.**
- **Clever choices lead to polynomial algorithms.**

Goal: choose augmenting paths so that:

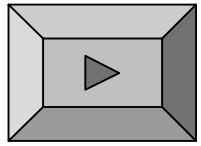
- **Can find augmenting paths efficiently.**
- **Few iterations.**

Edmonds-Karp (1972): choose augmenting path with

- **Max bottleneck capacity.** **(fat path)**
- **Sufficiently large capacity.** **(capacity-scaling)**
- **Fewest number of arcs.** **(shortest path)**

Shortest Augmenting Path

Intuition: choosing path via breadth first search.



- **Easy to implement.**
  - **may implement by coincidence!**
- **Finds augmenting path with fewest number of arcs.**

ShortestAugmentingPath(V, E, s, t)

```
FOREACH e ∈ E
   f(e) ← 0
G_f ← residual graph

WHILE (there exists augmenting path)
   find such a path P by BFS
   f ← augment(f, P)
   update G_f
RETURN f
```

Shortest Augmenting Path: Overview of Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

- **Proof ahead.**

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

- **Proof ahead.**

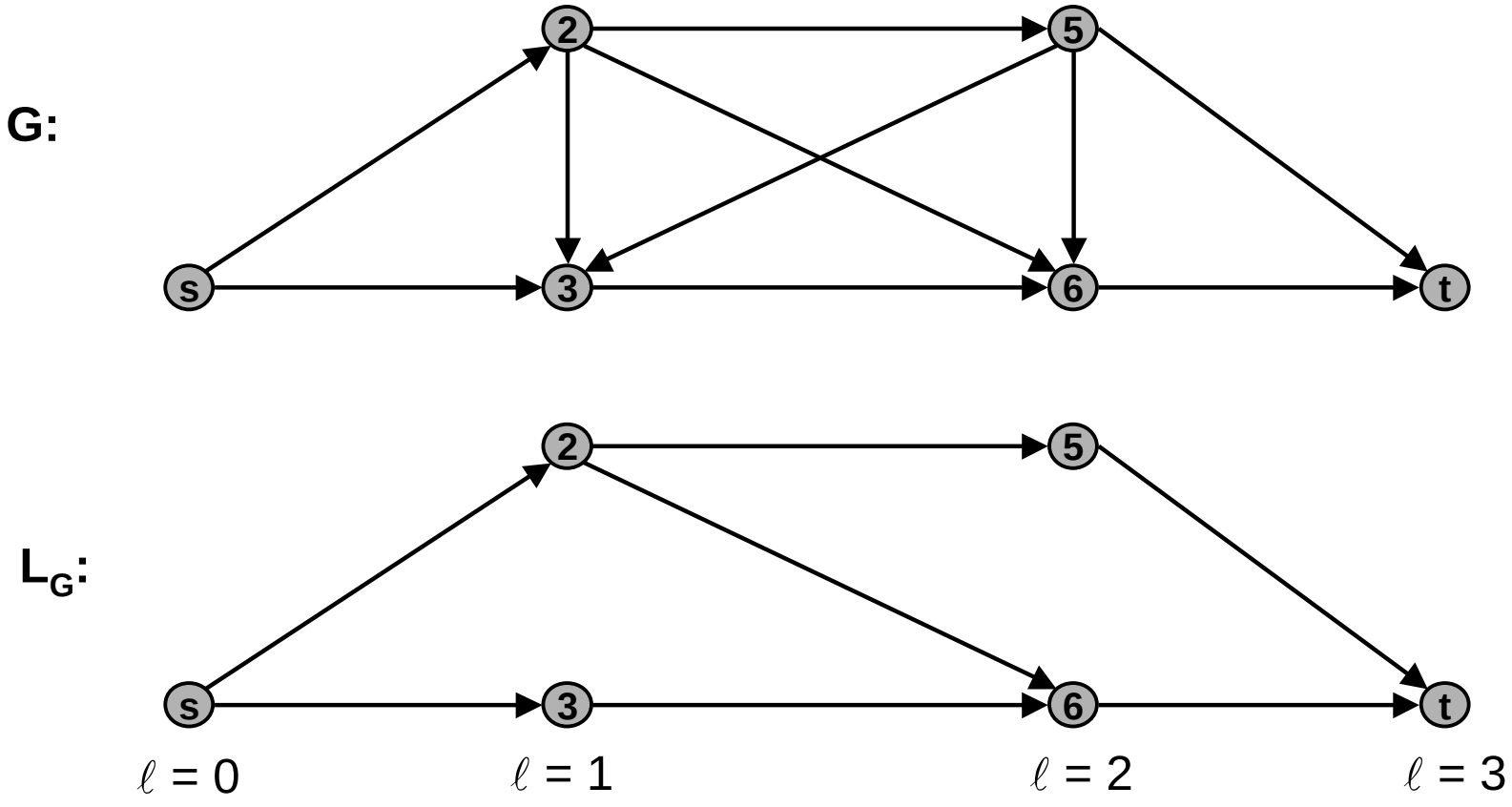
Theorem. The shortest augmenting path algorithm runs in $O(m^2n)$ time.

- **O(m+n) time to find shortest augmenting path via BFS.**
- **O(m) augmentations for paths of exactly k arcs.**
- **If there is an augmenting path, there is a simple one.**
  - $\Rightarrow$ **$1 \leq k < n$**
  - $\Rightarrow$ **O(mn) augmentations.**

Shortest Augmenting Path: Analysis

Level graph of (V, E, s).

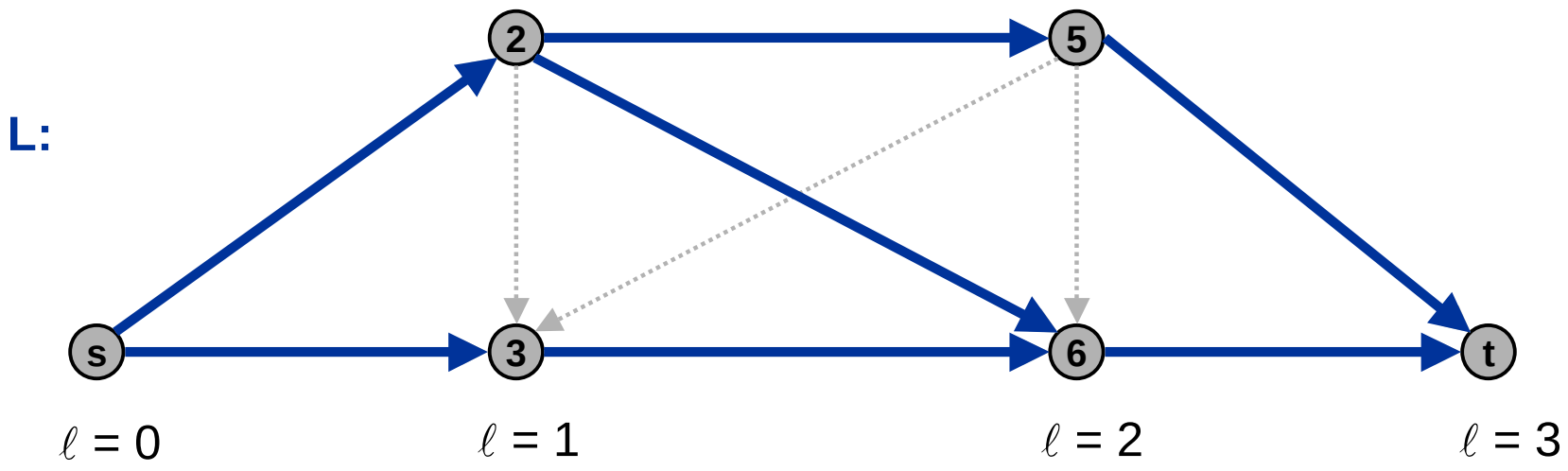
- **For each vertex v, define $\ell(v)$ to be the length (number of arcs) of shortest path from s to v.**
- **$L_G = (V, E_G)$ is subgraph of G that contains only those arcs $(v,w) \in E$ with $\ell(w) = \ell(v) + 1$.**



Shortest Augmenting Path: Analysis

Level graph of (V, E, s).

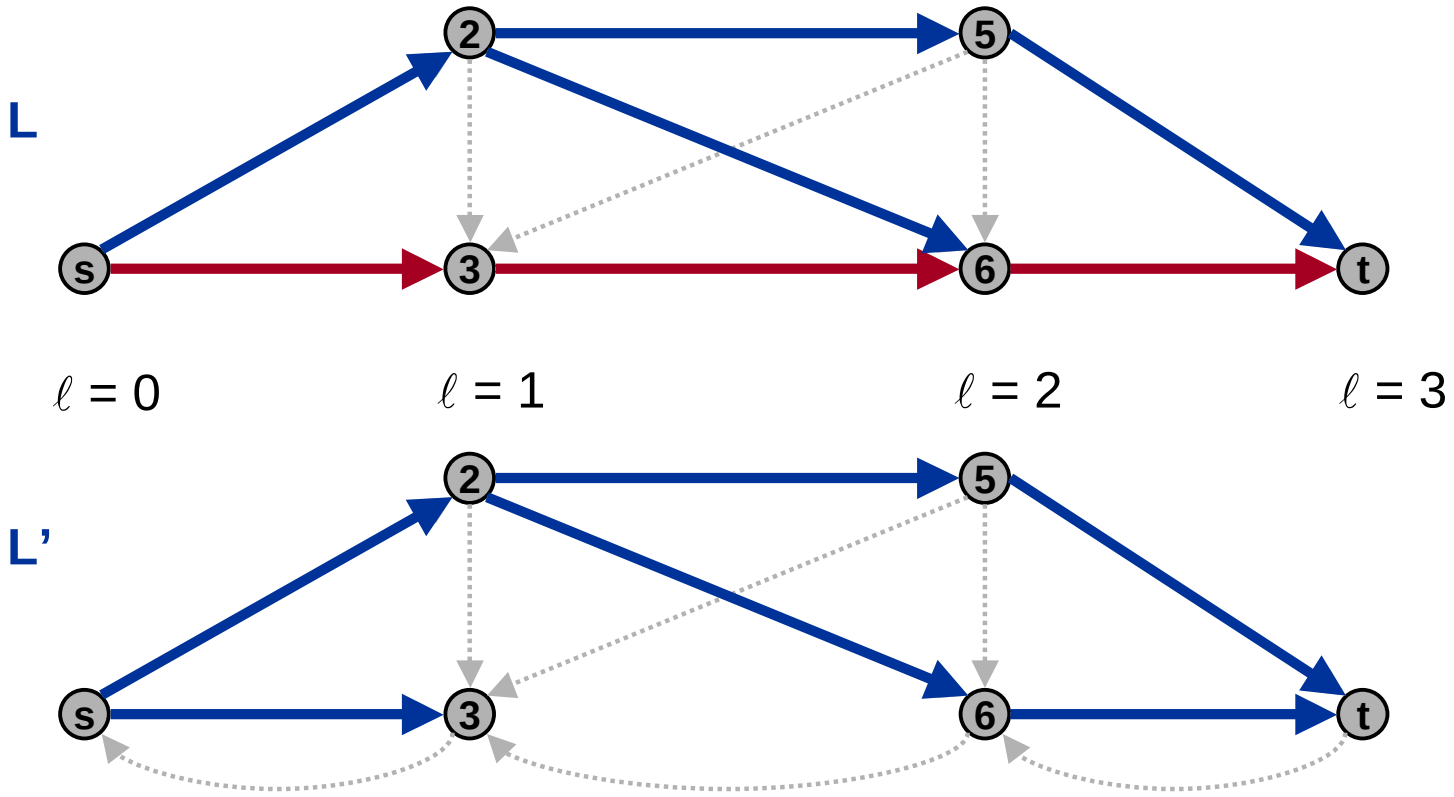
- **For each vertex v, define $\ell(v)$ to be the length (number of arcs) of shortest path from s to v.**
- **L = (V, F) is subgraph of G that contains only those arcs (v,w) $\in$ E with $\ell(w) = \ell(v) + 1$.**
- **Compute in O(m+n) time using BFS, deleting back and side arcs.**
- **P is a shortest s-v path in G if and only if it is an s-v path L.**



Shortest Augmenting Path: Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

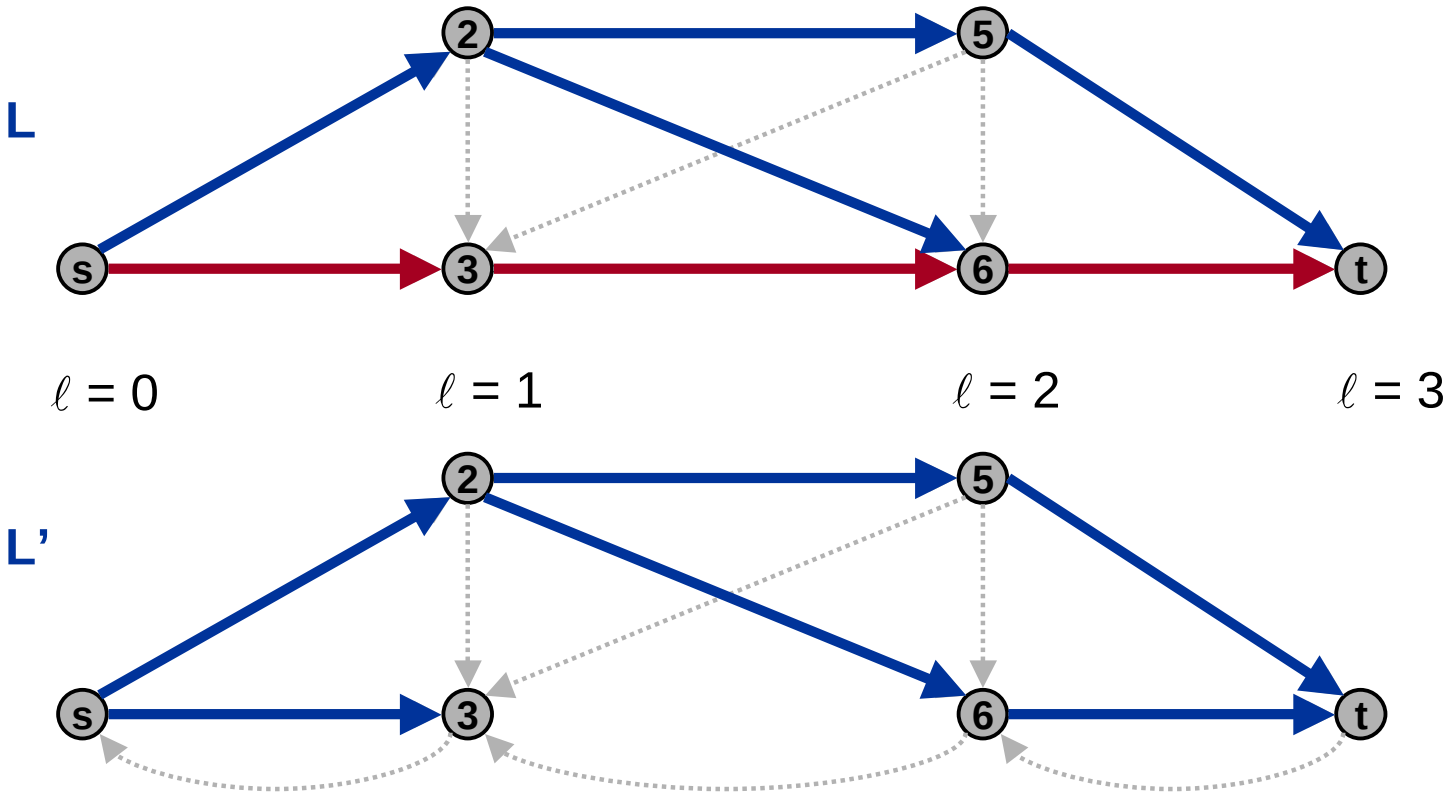
- **Let f and f' be flow before and after a shortest path augmentation.**
- **Let L and L' be level graphs of G_f and $G_{f'}$.**
- **Only back arcs added to $G_{f'}$.**
  - **path with back arc has length greater than previous length**



Shortest Augmenting Path: Analysis

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

- **At least one arc (the bottleneck arc) is deleted from L after each augmentation.**
- **No new arcs added to L until length of shortest path strictly increases.**



Shortest Augmenting Path: Review of Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

Theorem. The shortest augmenting path algorithm runs in $O(m^2n)$ time.

- **O(m+n) time to find shortest augmenting path via BFS.**
- **O(m) augmentations for paths of exactly k arcs.**
- **O(mn) augmentations.**

Note: $\Theta(mn)$ augmentations necessary on some networks.

- **Try to decrease time per augmentation instead.**
- **Dynamic trees $\Rightarrow$ O(mn log n)** Sleator-Tarjan, 1983
- **Simple idea $\Rightarrow$ $O(mn^2)$**

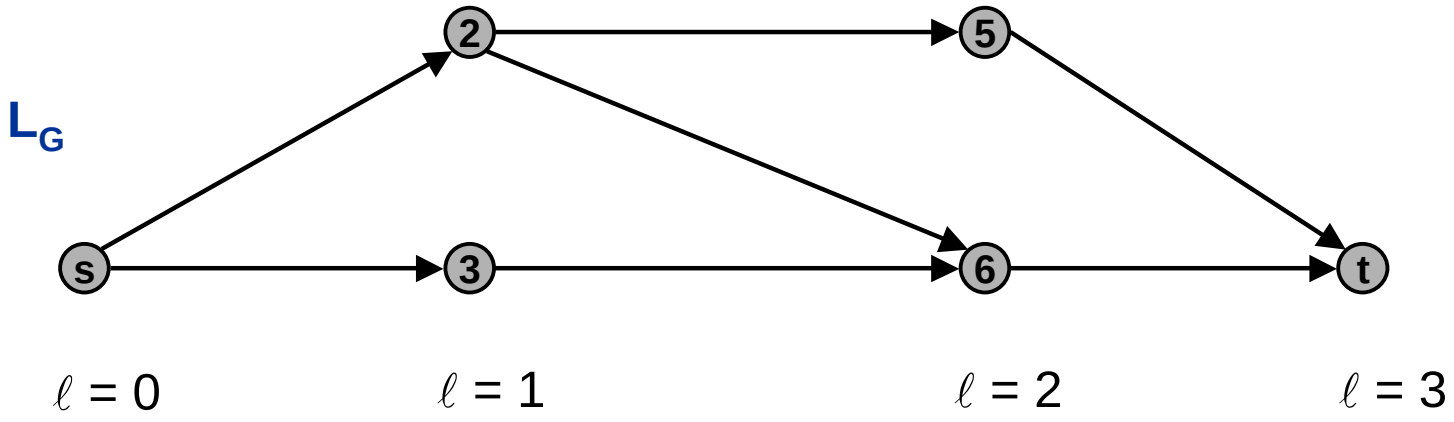
Shortest Augmenting Path: Improved Version

Two types of augmentations.

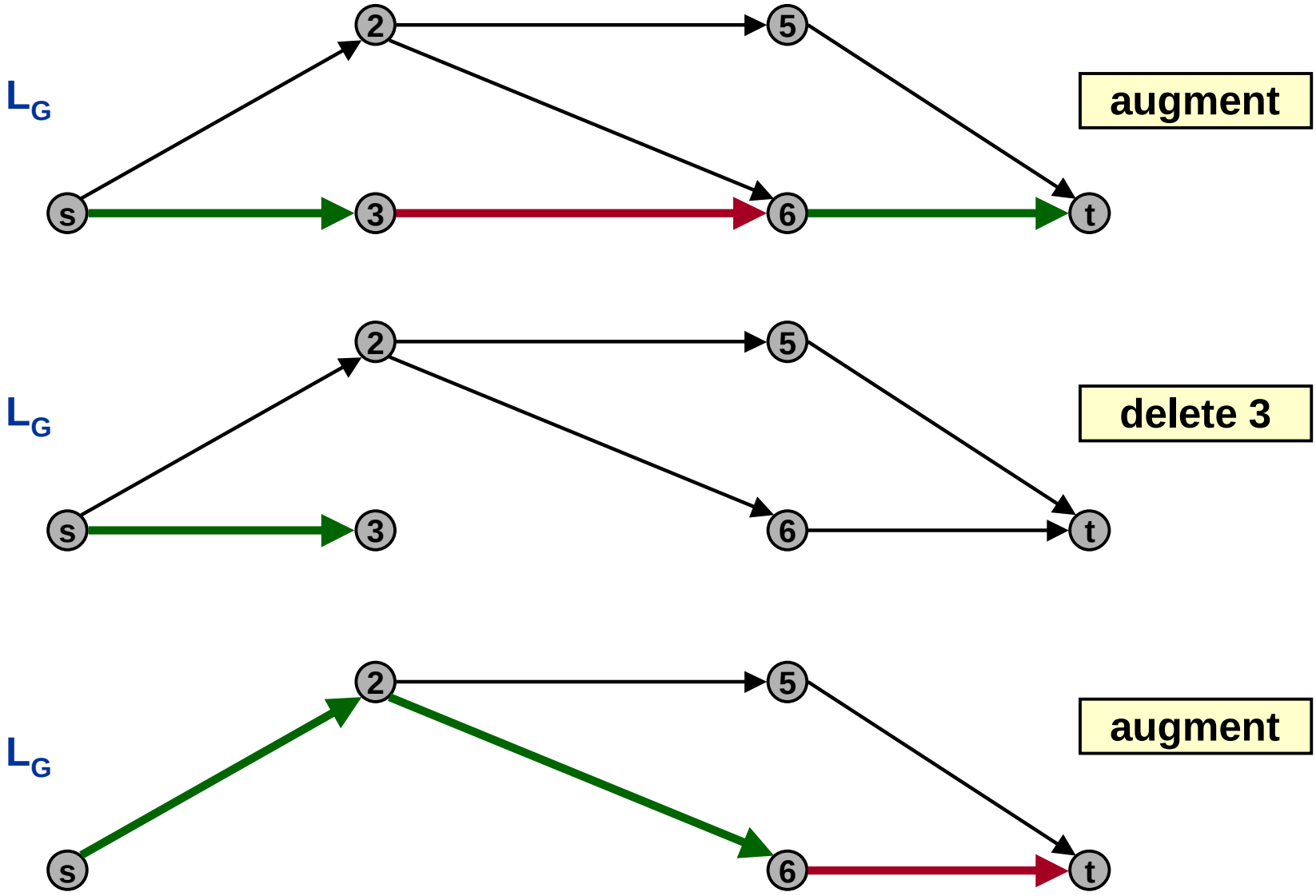
- **Normal augmentation: length of shortest path doesn't change.**
- **Special augmentation: length of shortest path strictly increases.**

L3. Group of normal augmentations takes O(mn) time.

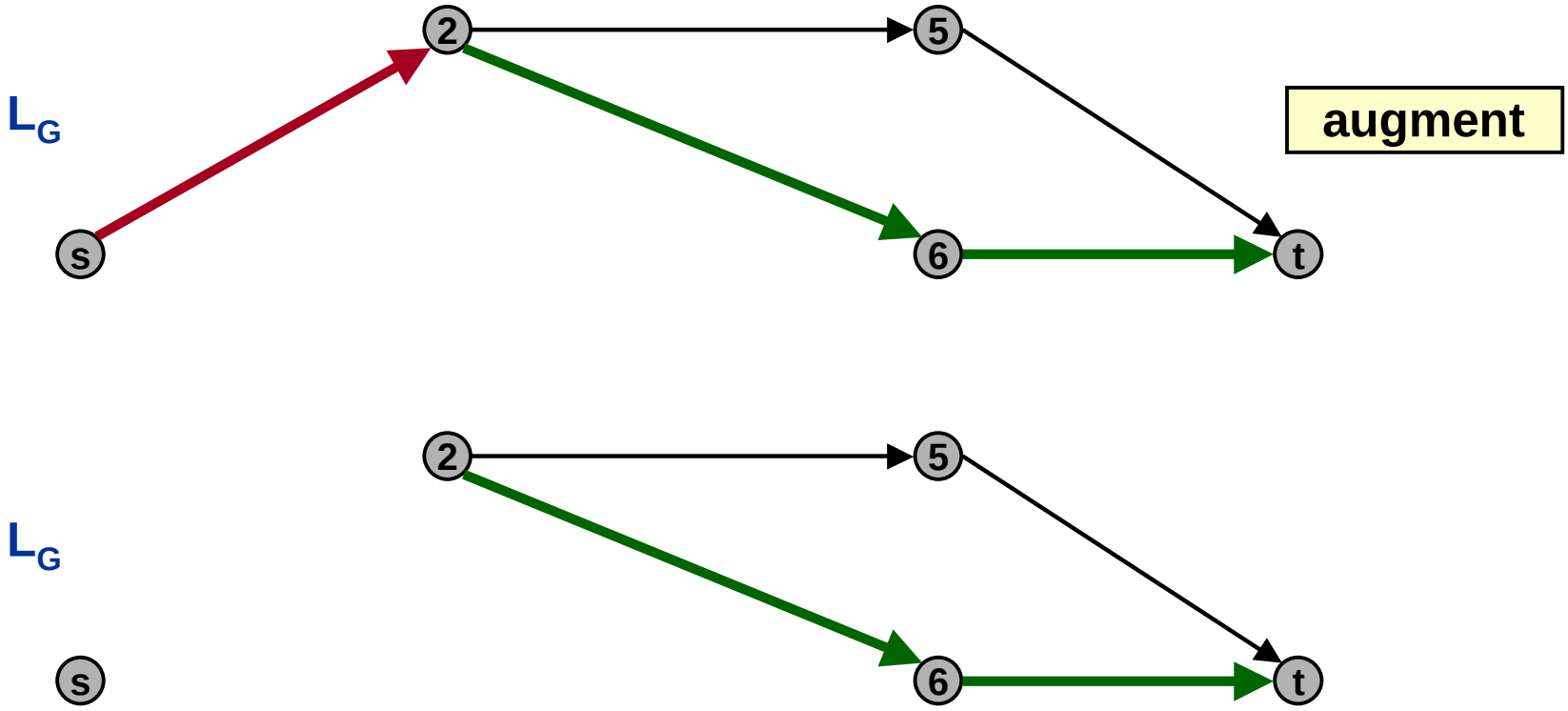
- **Explicitly maintain level graph - it changes by at most 2n arcs after each normal augmentation.**
- **Start at s, advance along an arc in L_G until reach t or get stuck.**
  - **if reach t, augment and delete at least one arc**
  - **if get stuck, delete node**



Shortest Augmenting Path: Improved Version



Shortest Augmenting Path: Improved Version



STOP: length of shortest path must have strictly increased

Shortest Augmenting Path: Improved Version

AdvanceRetreat(V, E, f, s, t)

```
ARRAY pred[v ∈ V]
L_G ← level graph of G_f
v ← s, pred[v] ← nil

REPEAT
   WHILE (there exists (v,w) ∈ L_G)
      pred[w] ← v, v ← w                    advance
      IF (v = t)
         P ← path defined by pred[]         augment
         f ← augment(f, P)
         update L_G
         v ← s, pred[v] ← nil
   delete v from L_G                        retreat
UNTIL (v = s)

RETURN f
```

Shortest Augmenting Path: Improved Version

Two types of augmentations.

- **Normal augmentation: length of shortest path doesn't change.**
- **Special augmentation: length of shortest path strictly increases.**

L3. Group of normal augmentations takes O(mn) time.

- **Explicitly maintain level graph - it changes by at most 2n arcs after each normal augmentation.**
- **Start at s, advance along an arc in L_G until reach t or get stuck.**
  - **if reach t, augment and delete at least one arc**
  - **if get stuck, delete node**
  - **at most n advance steps before one of above events**

Theorem. Algorithm runs in $O(mn^2)$ time.

- **O(mn) time between special augmentations.**
- **At most n special augmentations.**

Choosing Good Augmenting Paths: Summary

	Method	Augmentations	Running time
➡	Augmenting path	nU	mnU
	Max capacity	m log U	m log U (m + n log n)
➡	Capacity scaling	m log U	m^2 log U
	Improved capacity scaling	m log U	mn log U
➡	Shortest path	mn	m^2n
➡	Improved shortest path	mn	mn^2

First 4 rules assume arc capacities are between 0 and U.

History

Year	Discoverer	Method	Big-Oh
1951	Dantzig	Simplex	mn^2U
1955	Ford, Fulkerson	Augmenting path	mnU
1970	Edmonds-Karp	Shortest path	m^2n
1970	Dinitz	Shortest path	mn^2
1972	Edmonds-Karp, Dinitz	Capacity scaling	$m^2 \log U$
1973	Dinitz-Gabow	Capacity scaling	$mn \log U$
1974	Karzanov	Preflow-push	n^3
1983	Sleator-Tarjan	Dynamic trees	$mn \log n$
1986	Goldberg-Tarjan	FIFO preflow-push	$mn \log (n^2 / m)$
. . .	. . .	. . .	. . .
1997	Goldberg-Rao	Length function	$m^{3/2} \log (n^2 / m) \log U$ $mn^{2/3} \log (n^2 / m) \log U$