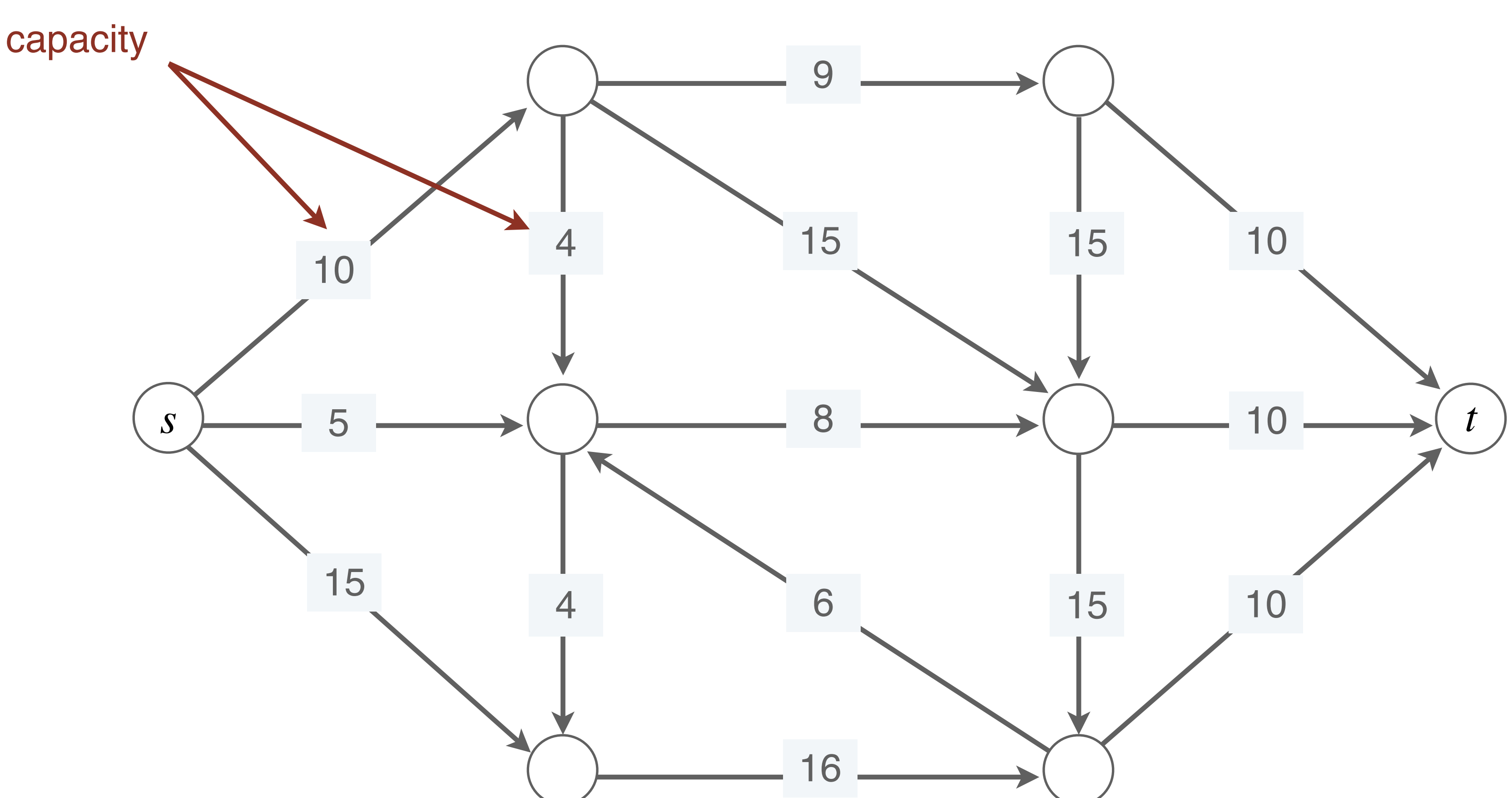
Network Flows

Admin

- Any questions before we start?

What's a Flow Network?

- A flow network is a directed graph $G = (V, E)$ with a
  - A **source** is a vertex s with in degree 0
  - A **sink** is a vertex t with out degree 0
  - Each edge $e \in E$ has **edge capacity** $c(e) > 0$



What's a Flow?

- Given a flow network, an (s, t)-flow or just flow (if source s and sink t are clear from context) $f: E \to \mathbb{Z}^+$ satisfies:
- **[Flow conservation]** $f_{in}(v) = f_{out}(v)$, for $v \neq s, t$ where

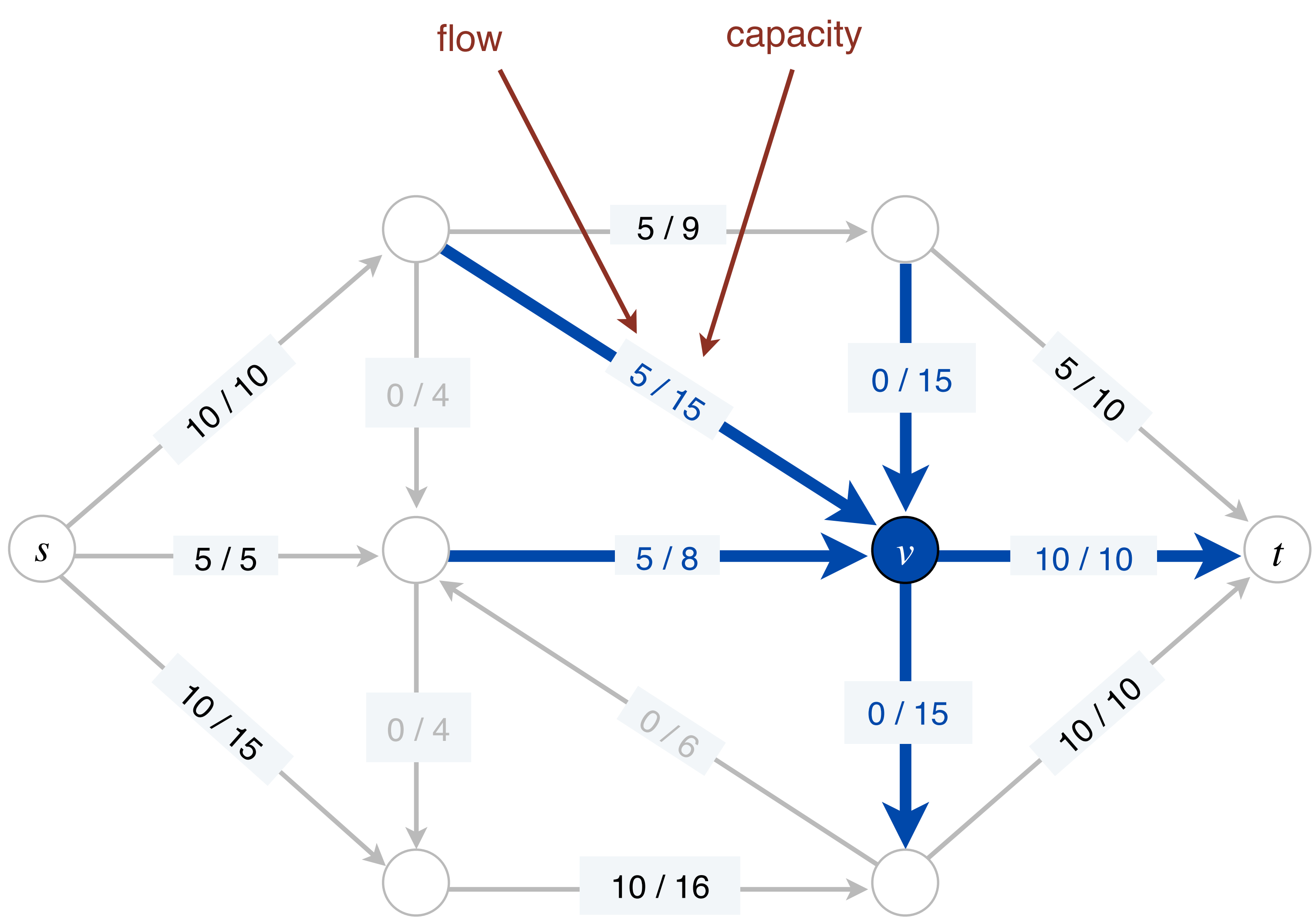
$$f_{in}(v) = \sum_u f(u \to v) \text{ and } f_{out}(v) = \sum_w f(v \to w)$$

- To simplify, $f(u \to v) = 0$ if there is no edge from u to v

What is a Feasible Flow

- An (s, t)-flow is **feasible** if it satisfies the capacity constraints of the network, that is,:

[Capacity constraint] for each $e \in E$, $0 \leq f(e) \leq c(e)$

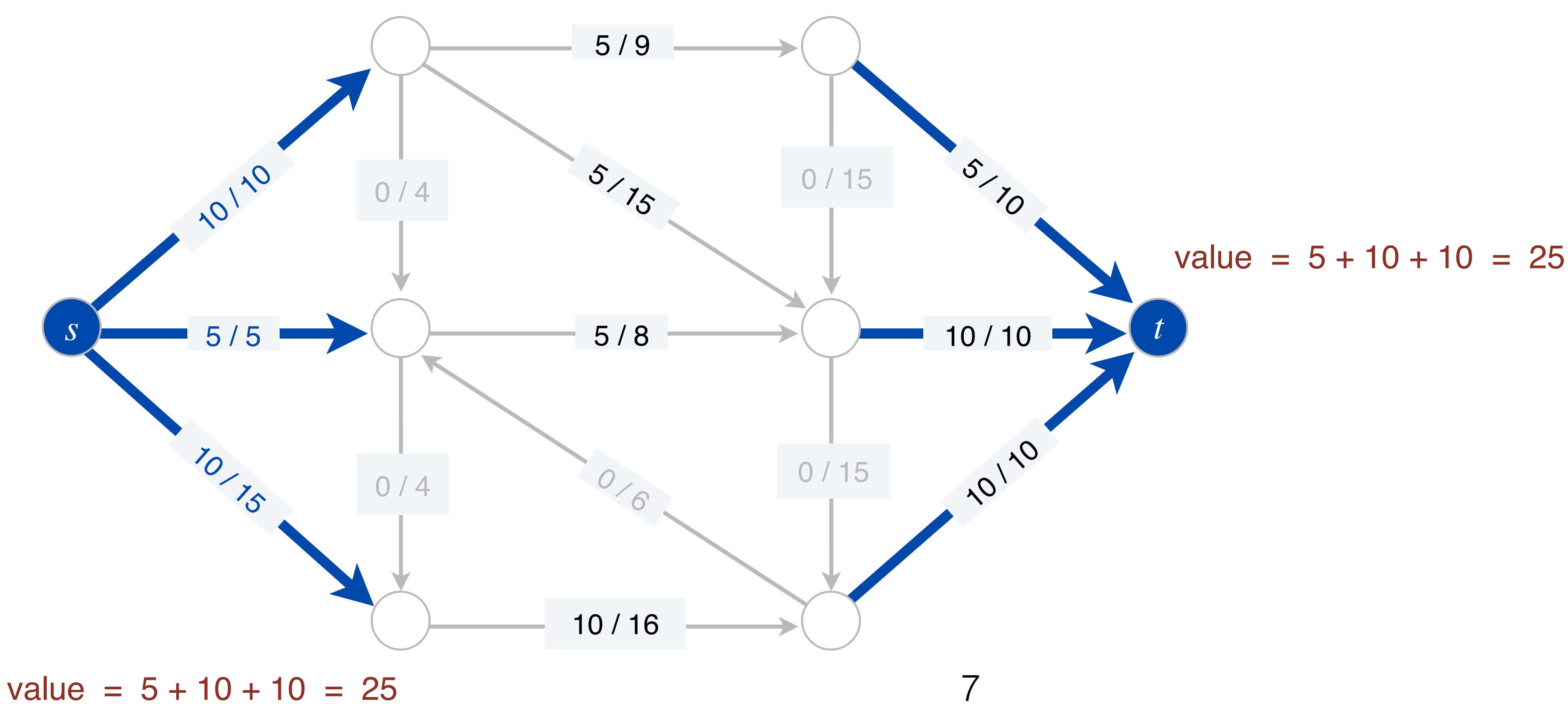


Value of a Flow

- **Definition.** The **value** of a flow f, written $v(f)$, is $f_{out}(s)$.
- **Lemma**. $f_{out}(s) = f_{in}(t)$
- **Proof**. Let $f(E) = \sum_{e \in E} f(e)$
  - Then, $\sum_{v \in V} f_{in}(v) = f(E) = \sum_{v \in V} f_{out}(v)$
  - For every $v \neq s, t$ flow conversation implies $f_{in}(v) = f_{out}(v)$
  - Thus all terms cancel out on both sides except $f_{in}(s) + f_{in}(t) = f_{out}(s) + f_{out}(t)$
  - But $f_{in}(s) = f_{out}(t) = 0$ ∎

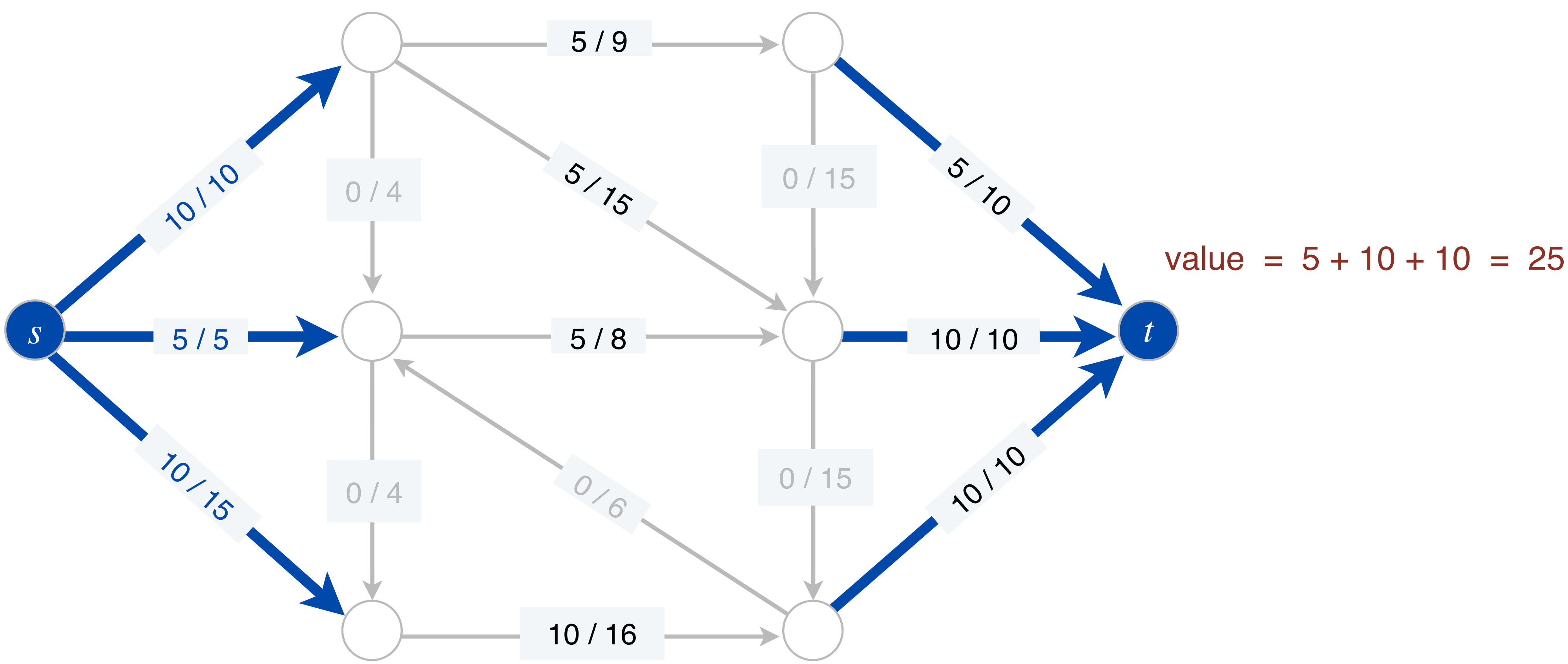
Value of a Flow

- **Definition.** The **value** of a flow f, written $v(f)$, is $f_{out}(s)$.
- **Lemma**. $f_{out}(s) = f_{in}(t)$
- **Corollary.** $v(f) = f_{in}(t)$.



Max-Flow Problem

- **Problem**. Given an s-t flow network, find a feasible s-t flow of maximum value.

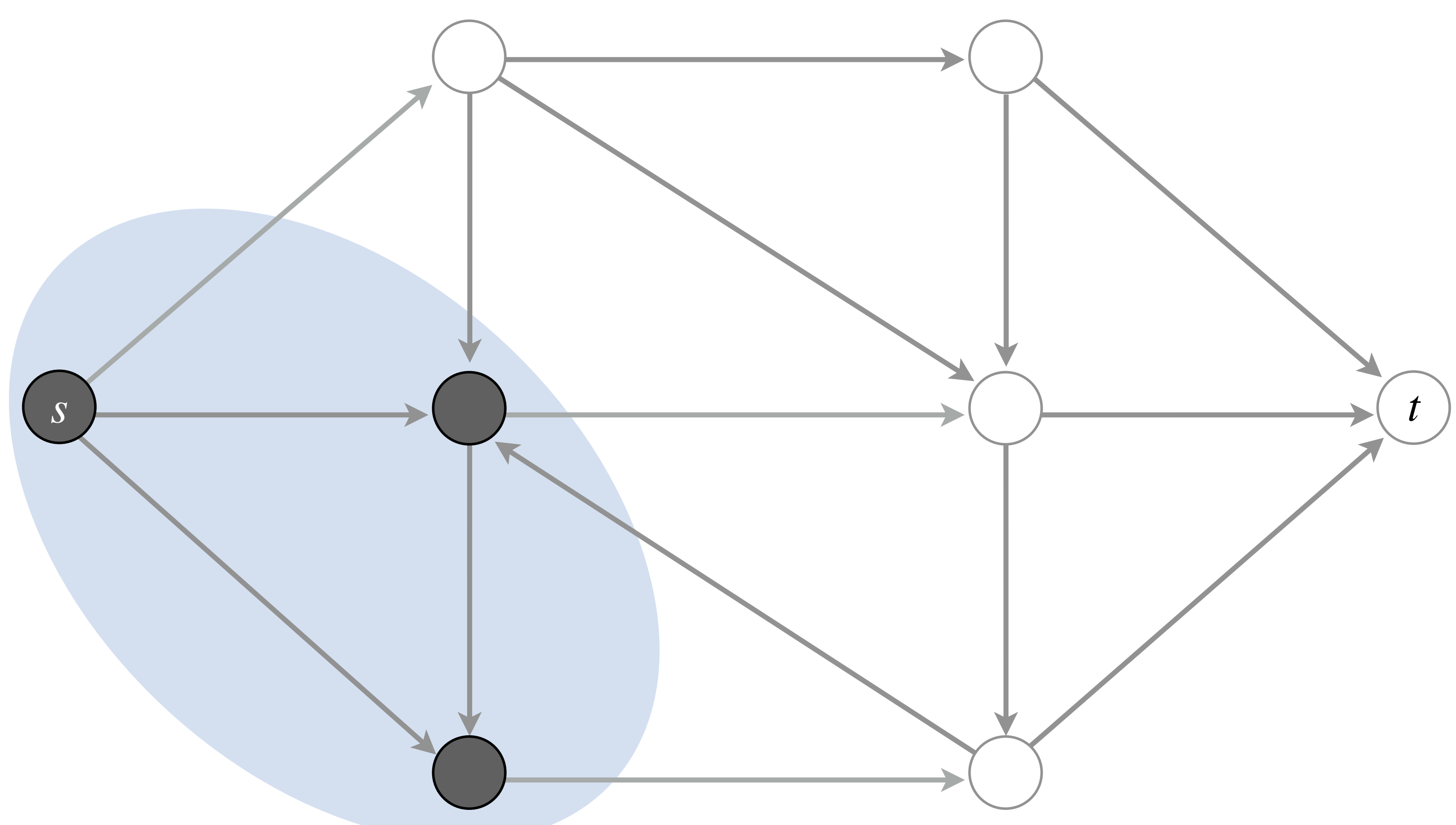


value = 5 + 10 + 10 = 25

Network Flows:
Max-Flow Min-Cut Theorem

Cuts in Flow Networks

- **Recall**. A cut (S, T) in a graph is a partition of vertices such that $S \cup T = V$, $S \cap T = \emptyset$ and S, T are non-empty.
- **Definition**. An (s, t)-*cut* is a cut (S, T) s.t. $s \in S$ and $t \in T$.



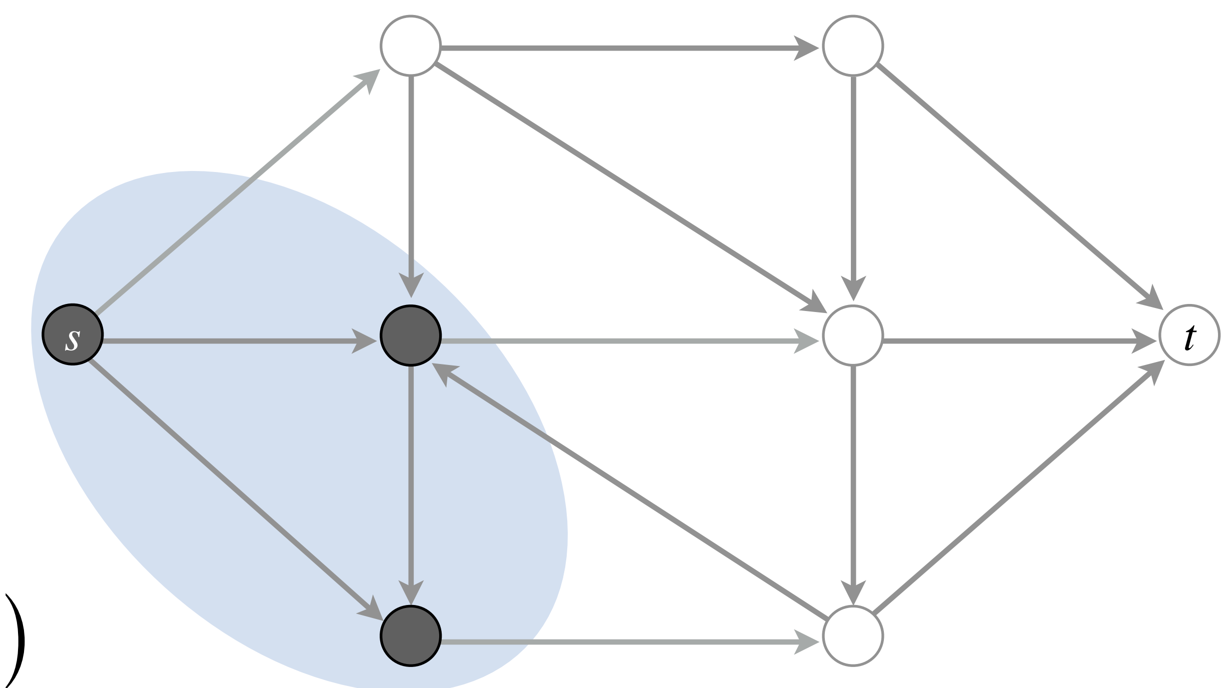
Cuts in Flow Networks

- For any flow f on $G = (V, E)$ and any (s, t)-*cut* (S, T), let
  - $f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w)$ (sum of flow 'leaving' S)
  - $f_{in}(S) = \sum_{v \in S, w \in T} f(w \to v)$ (sum of flow 'entering' S)
  - Note: $f_{out}(S) = f_{in}(T)$ and $f_{in}(S) = f_{out}(T)$
- **Lemma**. Value of a flow, $v(f) = f_{out}(S) - f_{in}(S)$ is the net-flow out of S, for any (s, t)-cut (S, T).

Value of Flow and Cuts

- **Lemma**. Value of a flow, $v(f) = f_{out}(S) - f_{in}(S)$ is the net-flow out of S, for any (s, t)-cut (S, T).

- **Proof**. $v(f) = f_{out}(s)$



$$v(f) = f_{out}(s) - f_{in}(s)$$

$$= \sum_{v \in S} (f_{out}(v) - f_{in}(v))$$

$$= \sum_{v \in S} \Big(\sum_{w} f(v \to w) - \sum_{u} f(u \to v) \Big)$$

$$= \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$$

$$= f_{out}(S) - f_{in}(S)$$

Value of Flow and Cuts (Explanation)

$$\sum_{v \in S} \Big(\sum_{w} f(v \to w) - \sum_{u} f(u \to v) \Big)$$

$$= \sum_{v \in S} \Big(\sum_{w \in S} f(v \to w) + \sum_{w \in T} f(v \to w) - \sum_{u \in S} f(u \to v) - \sum_{u \in T} f(u \to v) \Big)$$

$$= \left[\sum_{v \in S} \left(\sum_{w \in S} f(v \to w) - \sum_{u \in S} f(u \to v) \right) \right] + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$$

$$= \left[\sum_{v, w \in S} f(v \to w) - \sum_{v, u \in S} f(u \to v) \right] + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$$

$$= \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)$$

These are the same sum: they sum the flow of all edges with both vertices in S

Cut Capacity

- Capacity of a (s,t)-cut (S,T) is the sum of the capacities of edges leaving S:
    - $$c(S,T) = \sum_{v \in S, w \in T} c(v \to w)$$

Quick Quiz

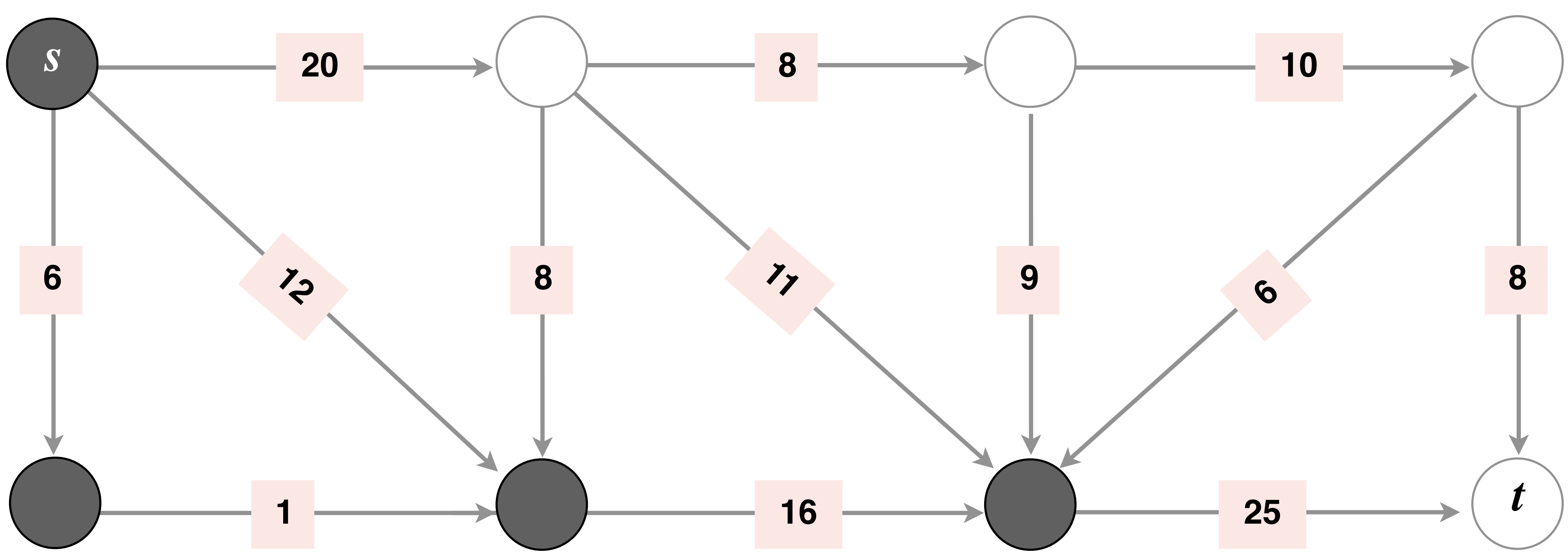
Question. Which is the capacity of the given st-cut?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 34 (8 + 11 + 9 + 6)

C. 45 (20 + 25)

D. 79 (20 + 25 + 8 + 11 + 9 + 6)



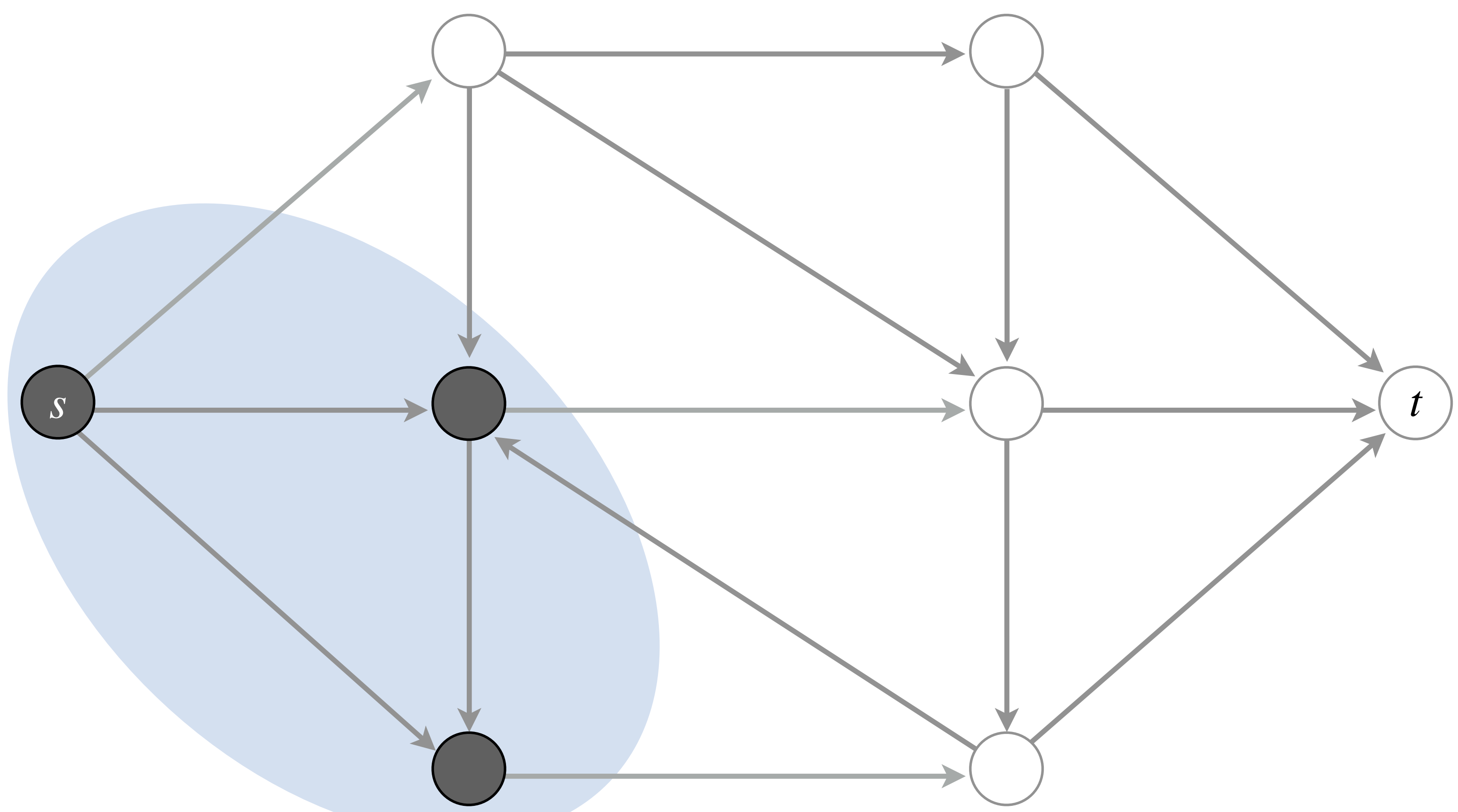
Capacities of Cuts

- Capacity of a (s, t)-cut (S, T) is the sum of the capacities of edges leaving S:
    - $$c(S, T) = \sum_{v \in S, w \in T} c(v \to w)$$
- **Dual problem of the max-flow problem**.
    - Find an (s, t)-cut of **minimum capacity**
- **Claim**. Let f be any s-t flow and (S, T) be any s-t cut then $v(f) \leq c(S, T)$

Cuts Upper Bound Flows

- For any cut, our flow needs to “get out” of that cut on its route from S to T

- So it seems the capacity of any cut is an upper limit on our max flow. Can we formalize that?



Relationship: Flows and Cuts

- **Claim**. Let f be any s-t flow and (S, T) be any s-t cut then $v(f) \leq c(S, T)$

- **Proof**.

    - $v(f) = f_{out}(S) - f_{in}(S)$

$$\leq f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w)$$

$$\leq \sum_{v \in S, w \in T} c(v, w) = c(S, T)$$

Max-Flow Min-Cut Theorem

- A beautiful, powerful relationship between these two problems in given by the following theorem

- **Theorem**. Given a flow network G, let f be an (s, t)-flow and let (S, T) be any (s, t)-cut of G then,

  $v(f) = c(S, T)$ **if and only if**

  f is a flow of maximum value and (S, T) is a cut of minimum capacity.

- Informally, in a flow network the max-flow = min-cut.

- (Will prove this theorem by construction in a bit.)

Max-Flow Problem Review

- **Max-flow problem**. Given a flow network $G = (V, E, c)$ with source s and sink t, find a feasible s-t flow of maximum value.
- Recall that a feasible flow must satisfy:
  - **Flow conservation**: $f_{in}(v) = f_{out}(v)$, for $v \neq s, t$ where

    $f_{in}(v) = \sum_u f(u \to v)$, and

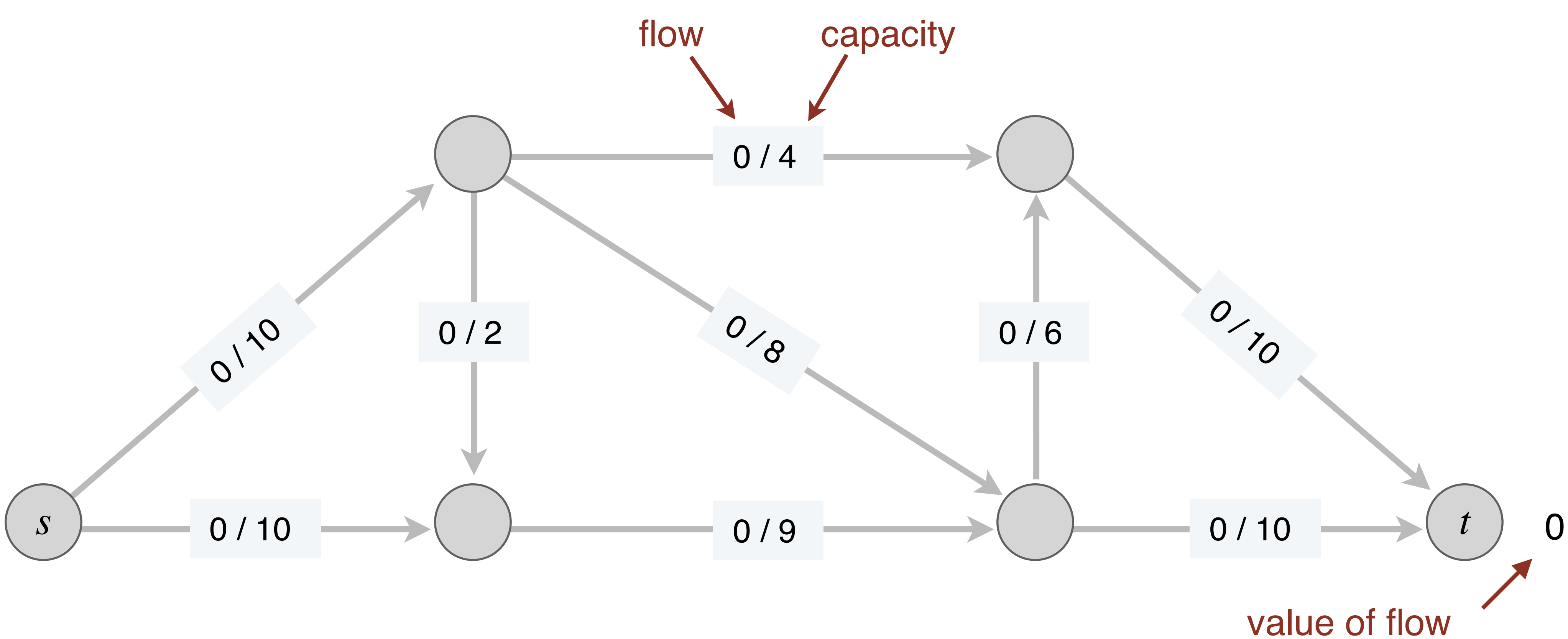
    $f_{out}(v) = \sum_w f(v \to w)$
  - **Capacity constraint**: for each $e \in E$, $0 \leq f(e) \leq c(e)$
- Recall that **the value of a flow** is $v(f) = f_{out}(s) = f_{in}(t)$.

Towards a Max-Flow Algorithm

- Greedy strategy:
  - Start with $f(e) = 0$ for each edge
  - Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
  - "Augment" flow (as much as possible) along path P
  - Repeat until you get stuck
- Let's take an example

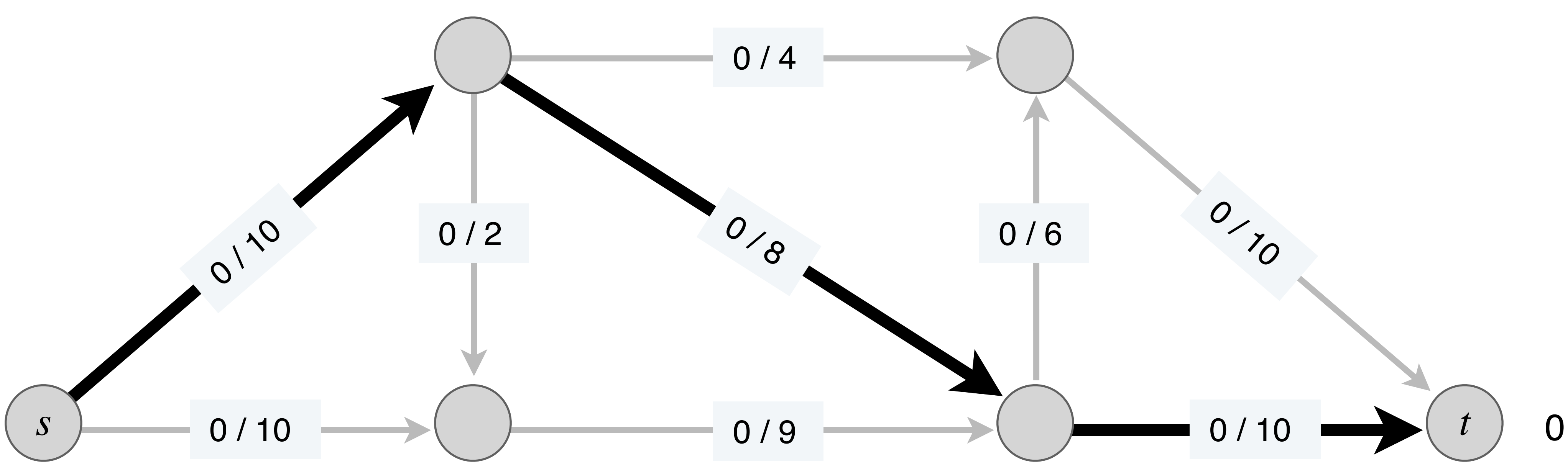
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



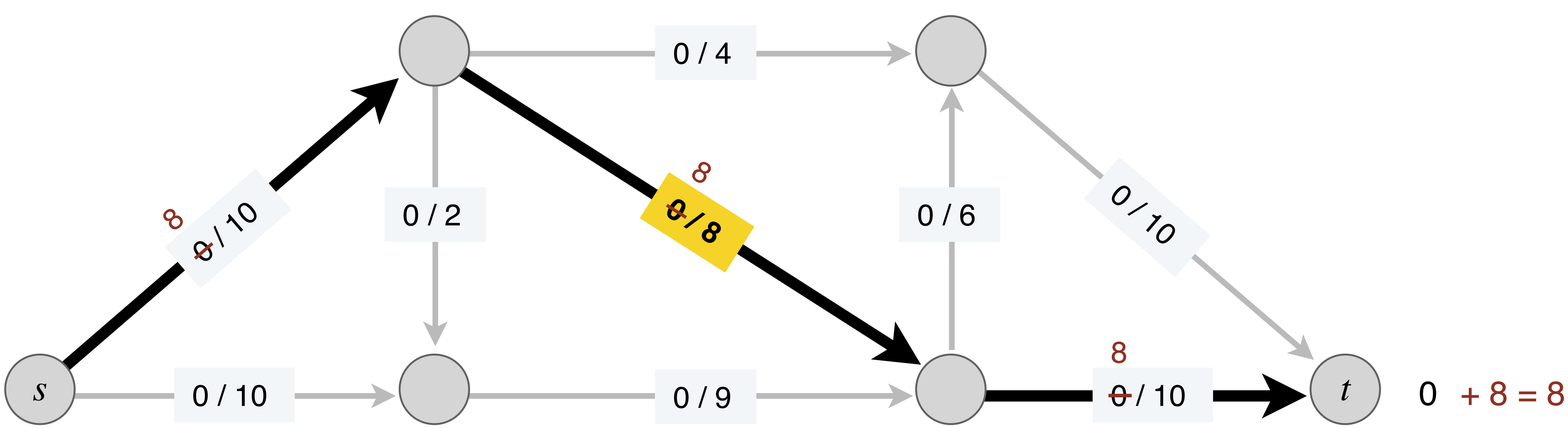
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path P
- Repeat until you get stuck



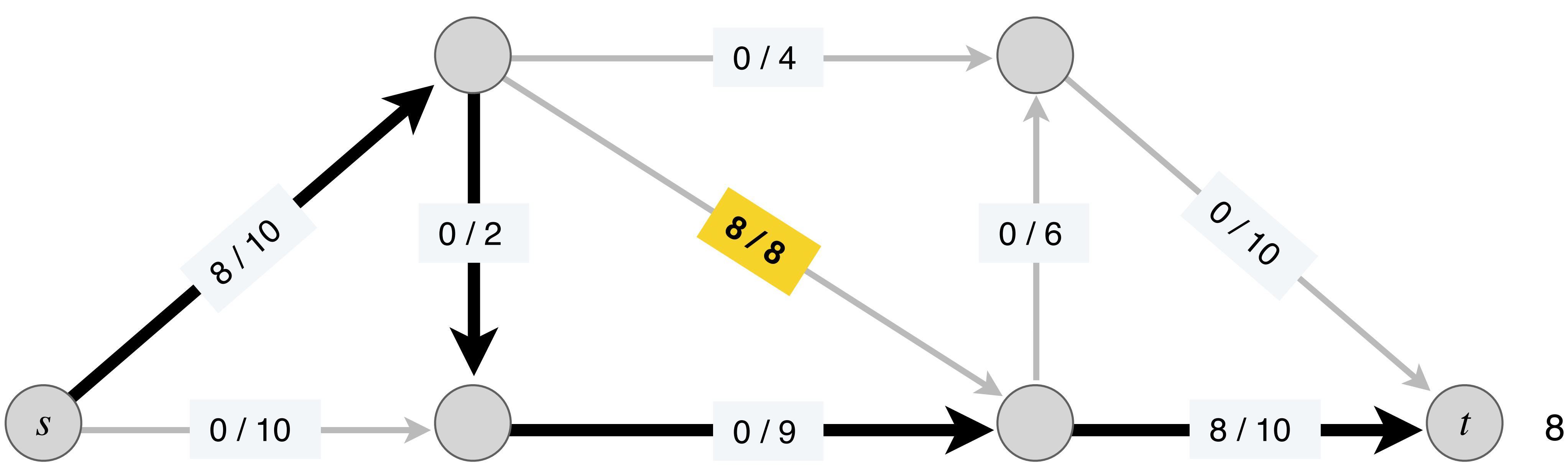
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



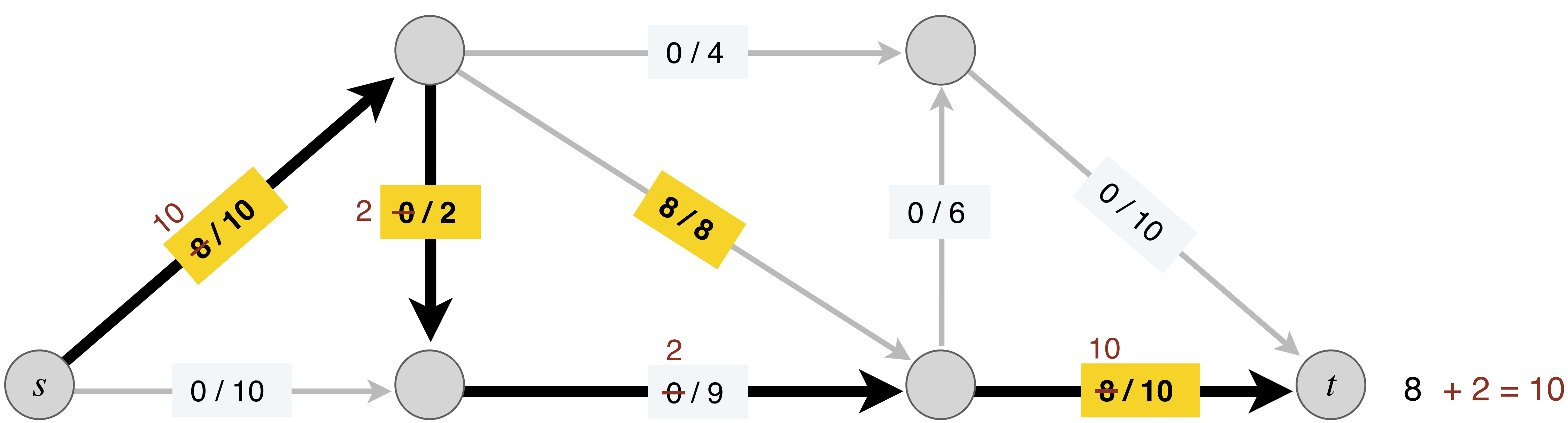
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



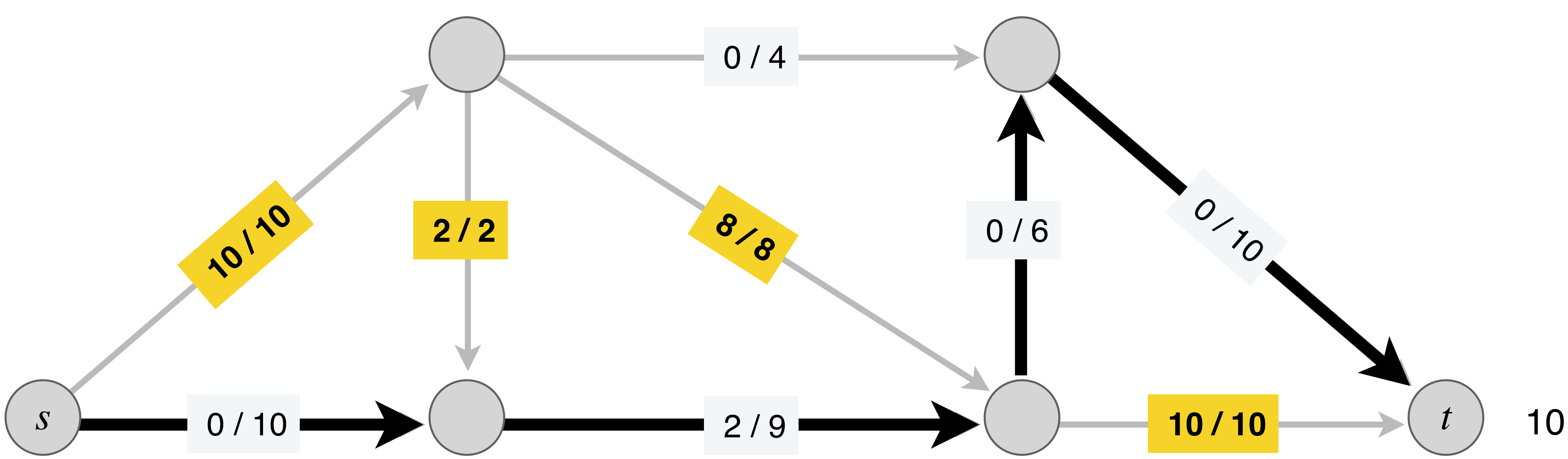
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



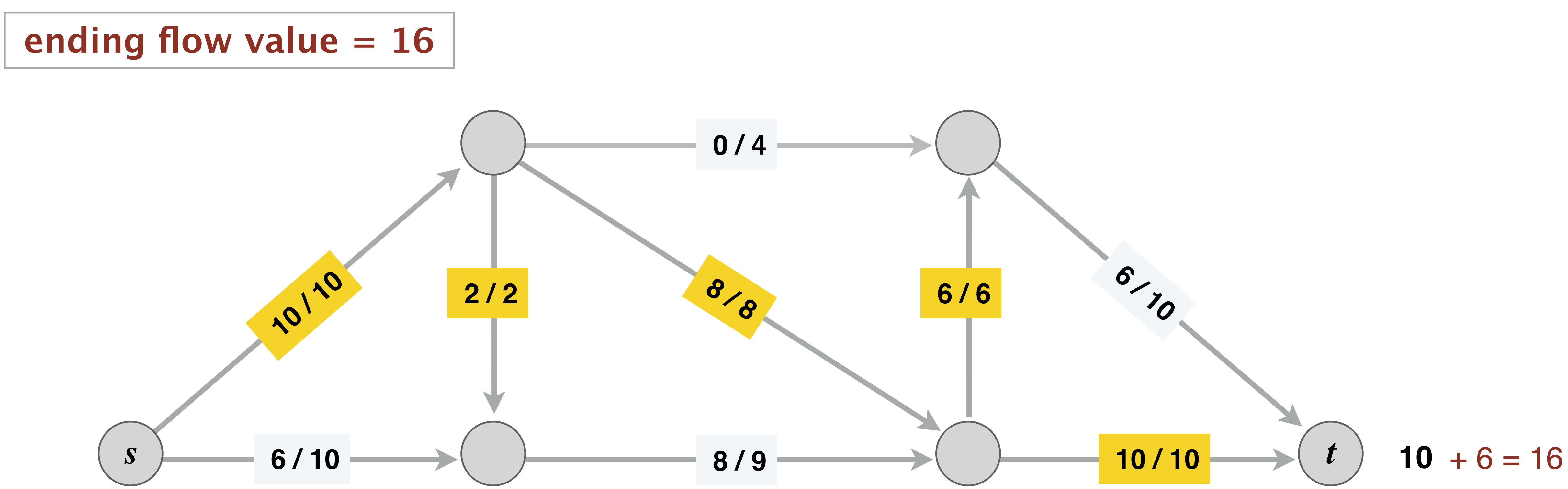
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



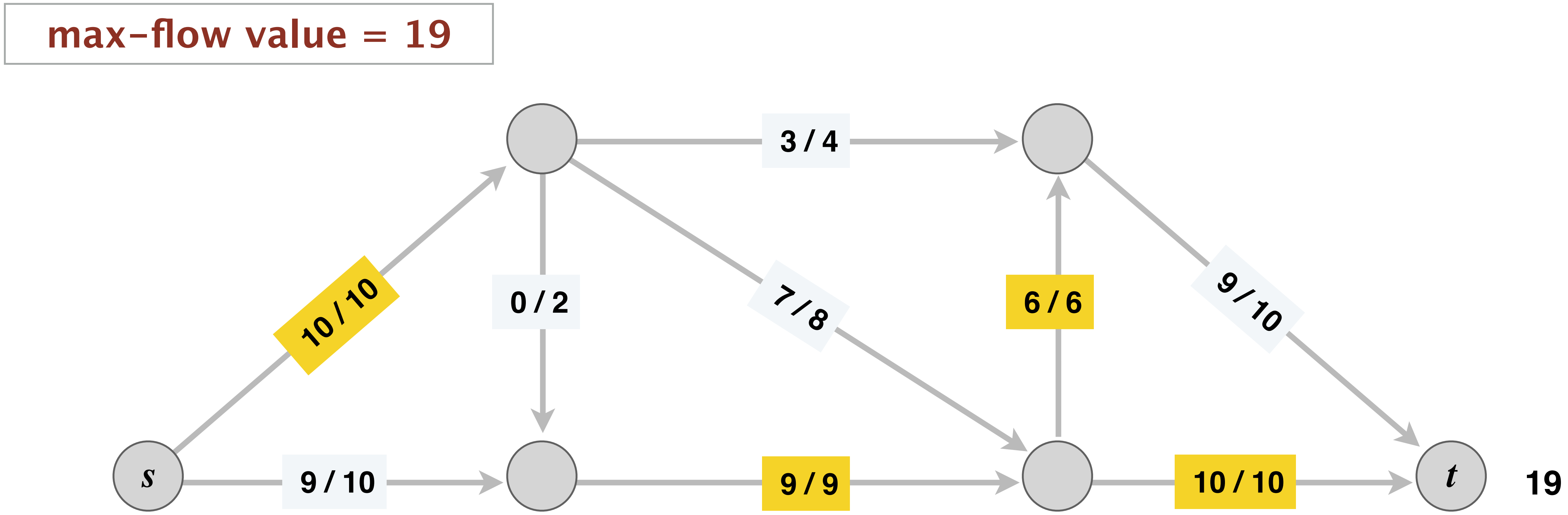
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



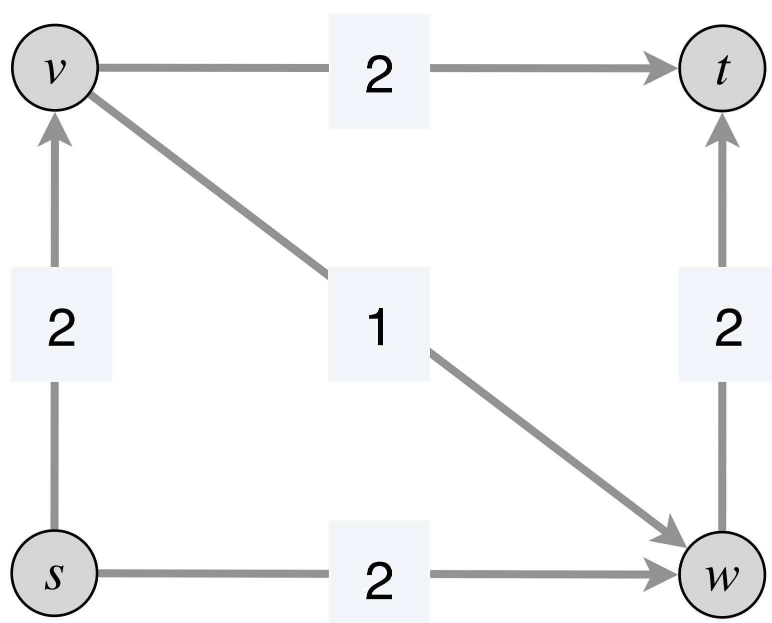
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path P
- Repeat until you get stuck



Why Greedy Fails

- Problem: greedy can never "undo" a bad flow decision
- Consider the following flow network
    - Unique max flow has $f(v \to w) = 0$
    - Greedy could choose $s \to v \to w \to t$ as first P



- **Summary**: Need a mechanism to "undo" bad flow decisions

Ford-Fulkerson Algorithm

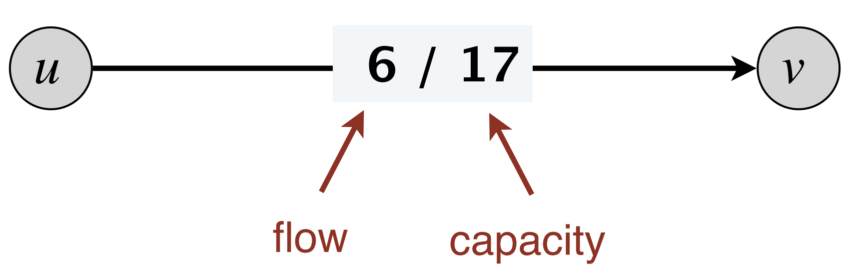
Ford Fulkerson: Idea

- We want to make “forward progress” while letting ourselves undo previous decisions if they’re getting in our way
- Idea: keep track of where we can push flow
  - Can push more flow along an edge with remaining capacity
  - Can also push flow “back” along an edge that already has flow down it

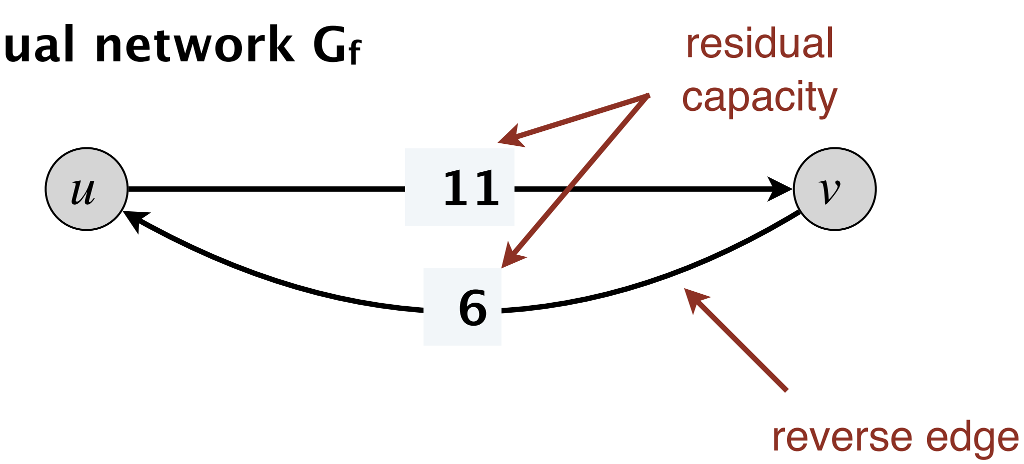
Residual Graph

- Given flow network $G = (V, E, c)$ and a feasible flow f on G, **the residual graph** $G_f = (V, E_f, c_f)$ is defined as:
  - Vertices in G_f same as G
  - **(Forward edge)** For $e \in E$ with residual capacity $c_r = c(e) - f(e) > 0$ let $e \in E_f$ with capacity c_r
  - **(Backward edge)** For $e \in E$ with $f(e) > 0$, let $e_{\text{reverse}} \in E_f$ with capacity $f(e)$

original flow network G



residual network G_f



Augmenting Path & Flow

- An **augmenting path** P is a simple $s \rightsquigarrow t$ path in the residual graph G_f
- The **bottleneck capacity** b of an augmenting path P is the minimum capacity of any edge in P.

AUGMENT(f, P)

$b \leftarrow$ bottleneck capacity of augmenting path P.

FOREACH edge $e \in P$:

  IF ($e \in E$, *that is, e is forward edge*)

    Increase f(e) in G by b

  ELSE

    Decrease f(e) in G by b

RETURN f.

Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for each edge $e \in E$
- Find an $s \leadsto t$ path P in the residual network G_f
- Augment flow along path P
- Repeat until you get stuck

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s⤳t path P in G_f)

  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

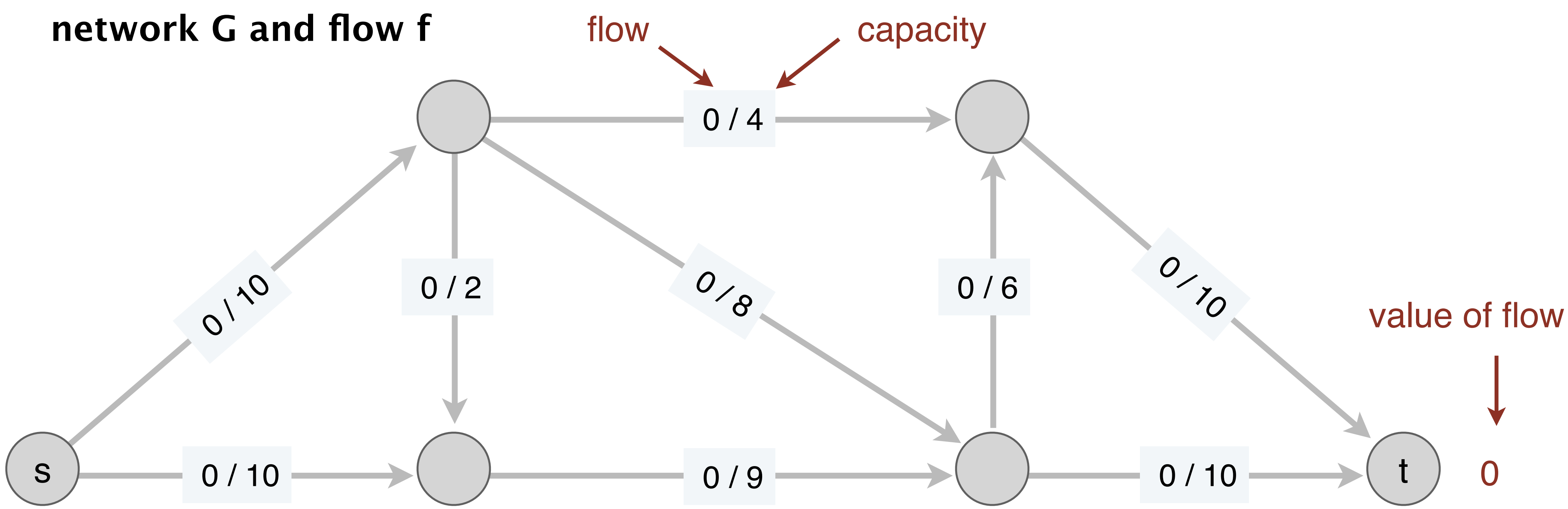
RETURN f.

Quick question

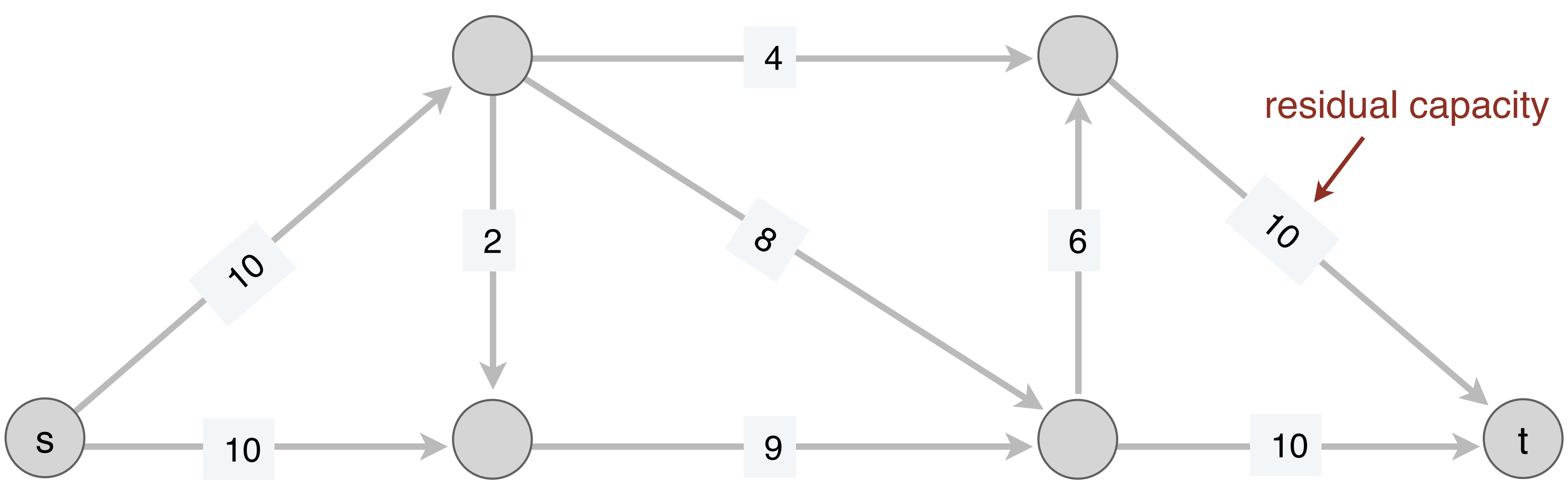
- Are we making forward progress here? (An augmenting path can "push flow back")
- Yes: cannot push flow back out of t—the last edge always involves more flow. So the augmenting path always increases the flow into t

Ford-Fulkerson Example

network G and flow f

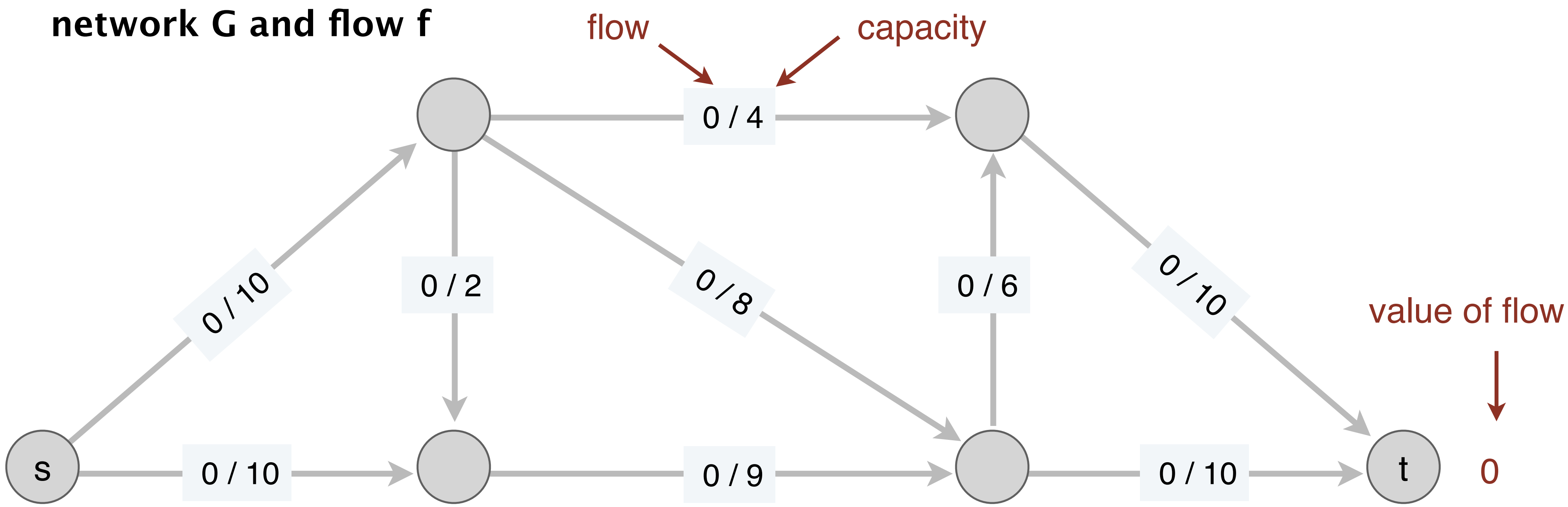


residual network G_f

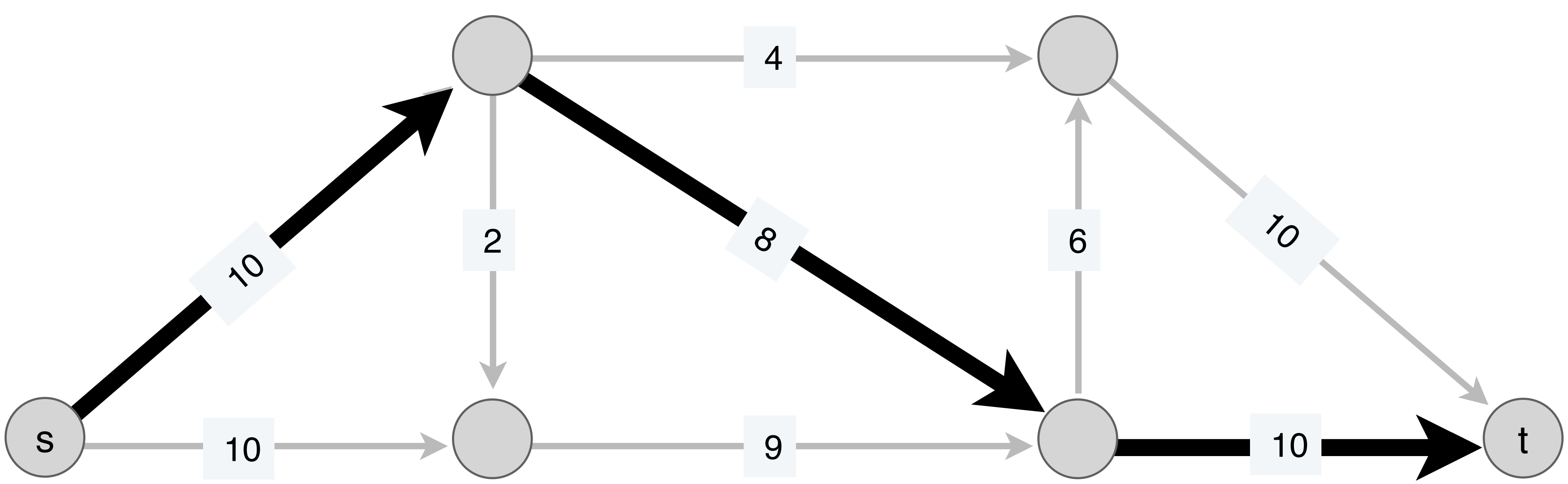


Ford-Fulkerson Example

network G and flow f

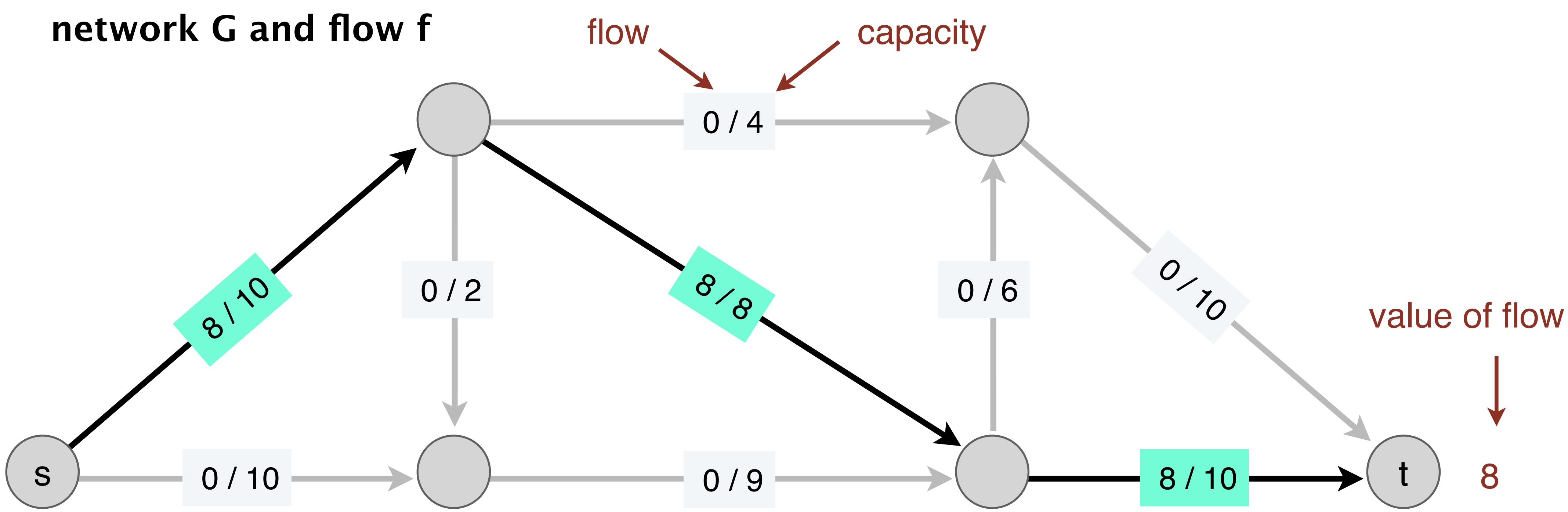


P in residual network G_f

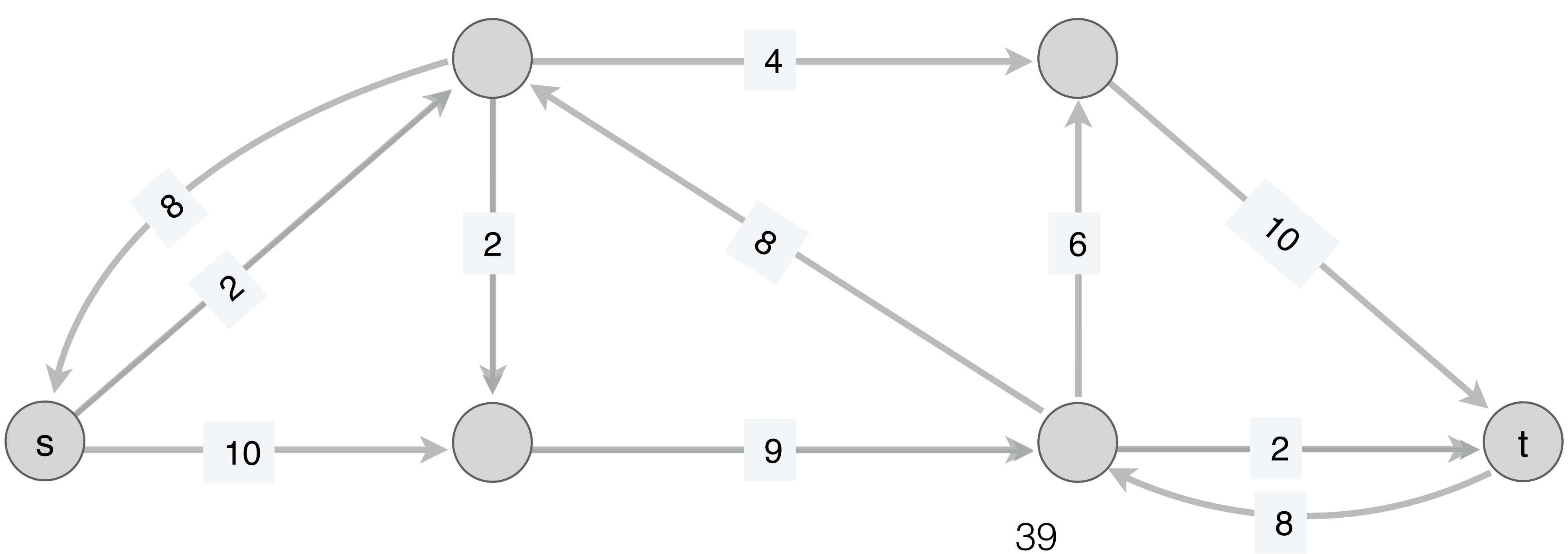


Ford-Fulkerson Example

network G and flow f

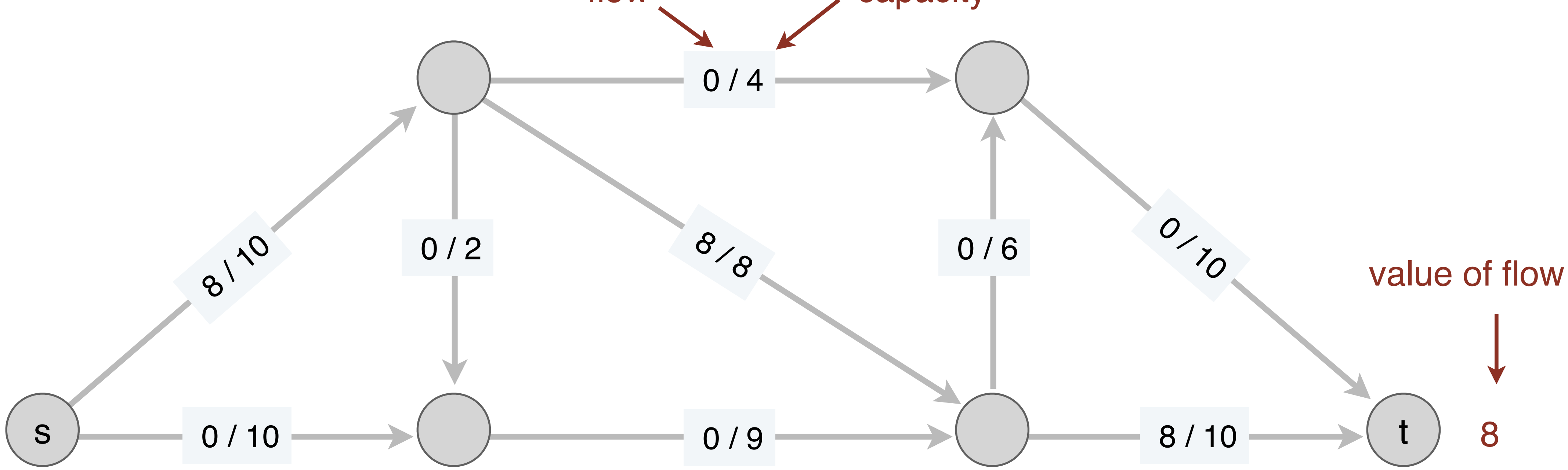


residual network G_f

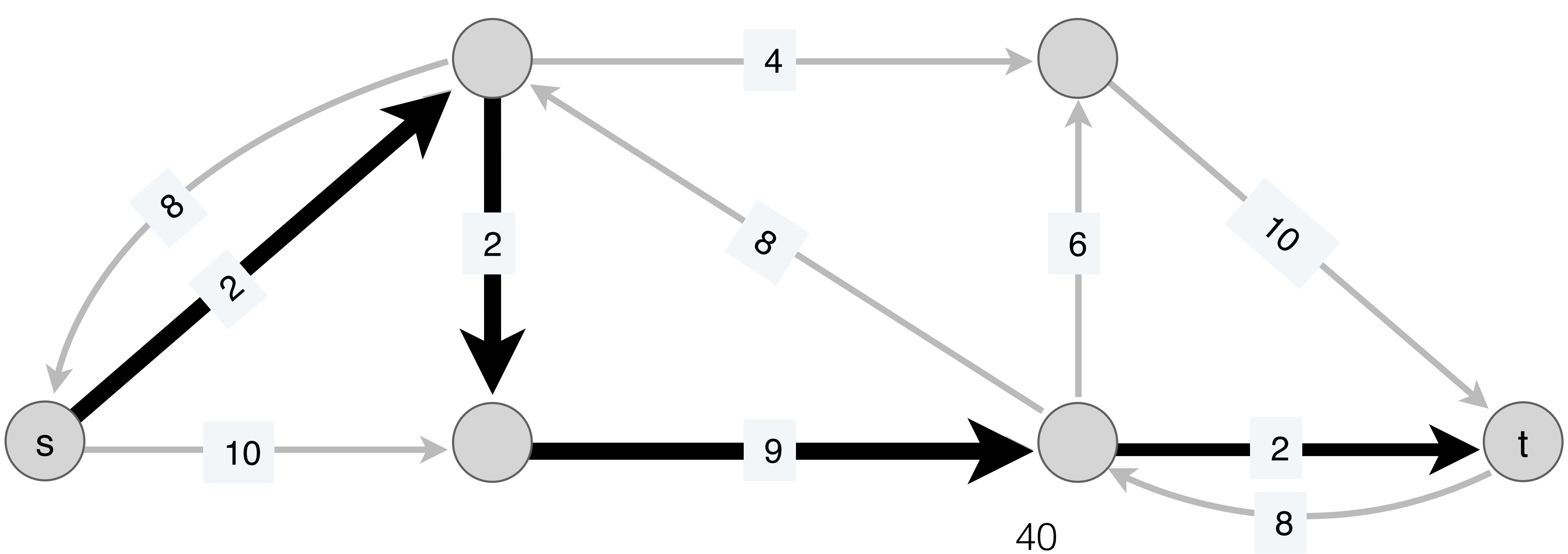


Ford-Fulkerson Example

network G and flow f

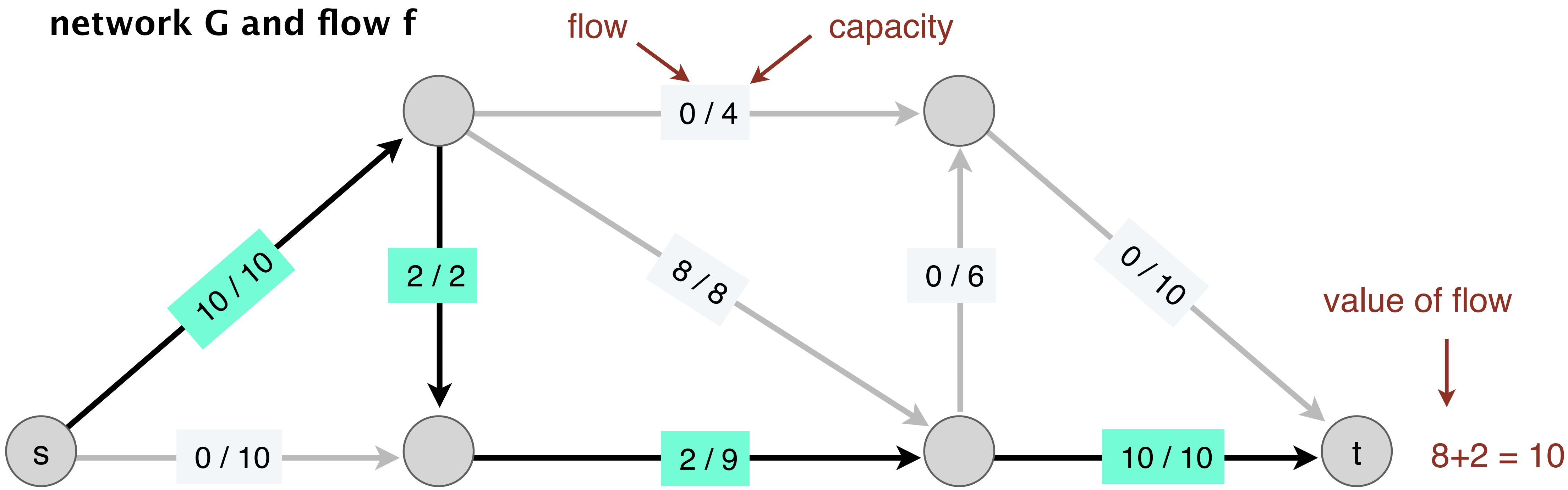


P in residual network G_f

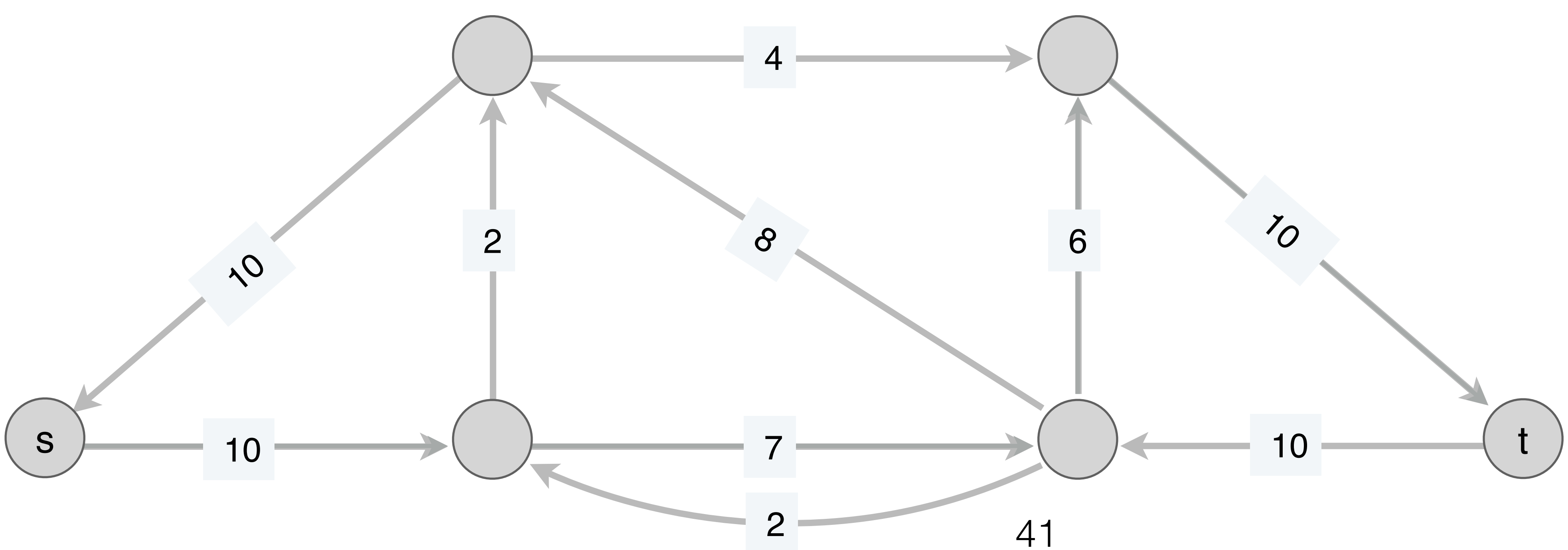


Ford-Fulkerson Example

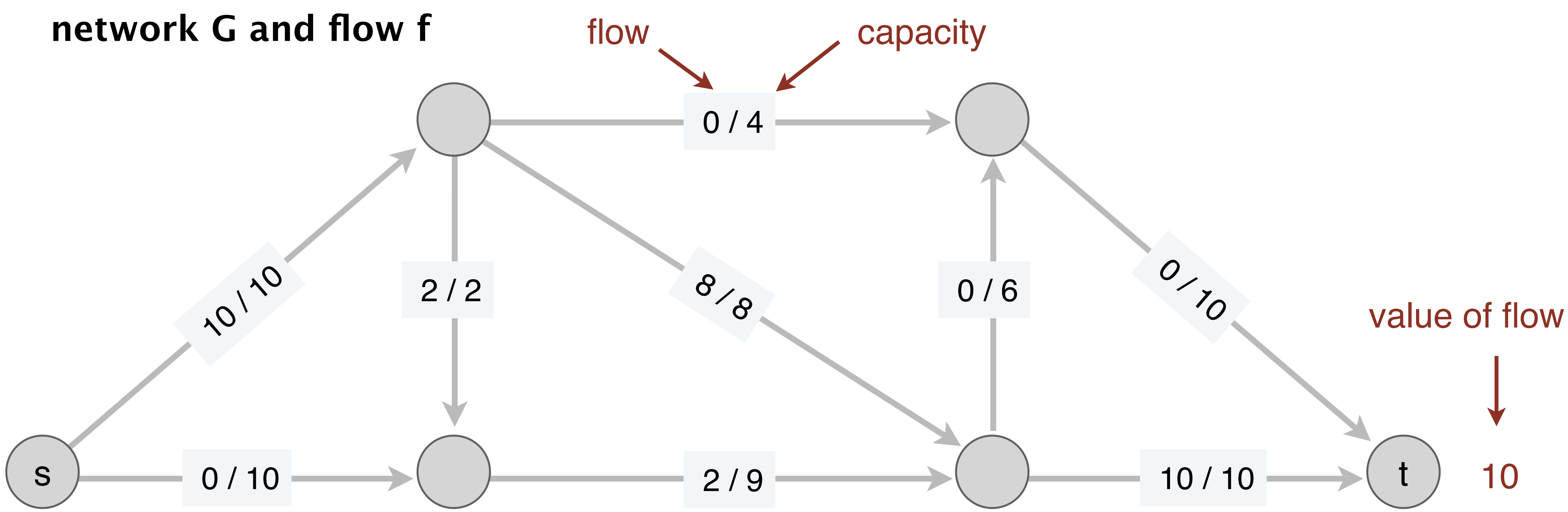
network G and flow f

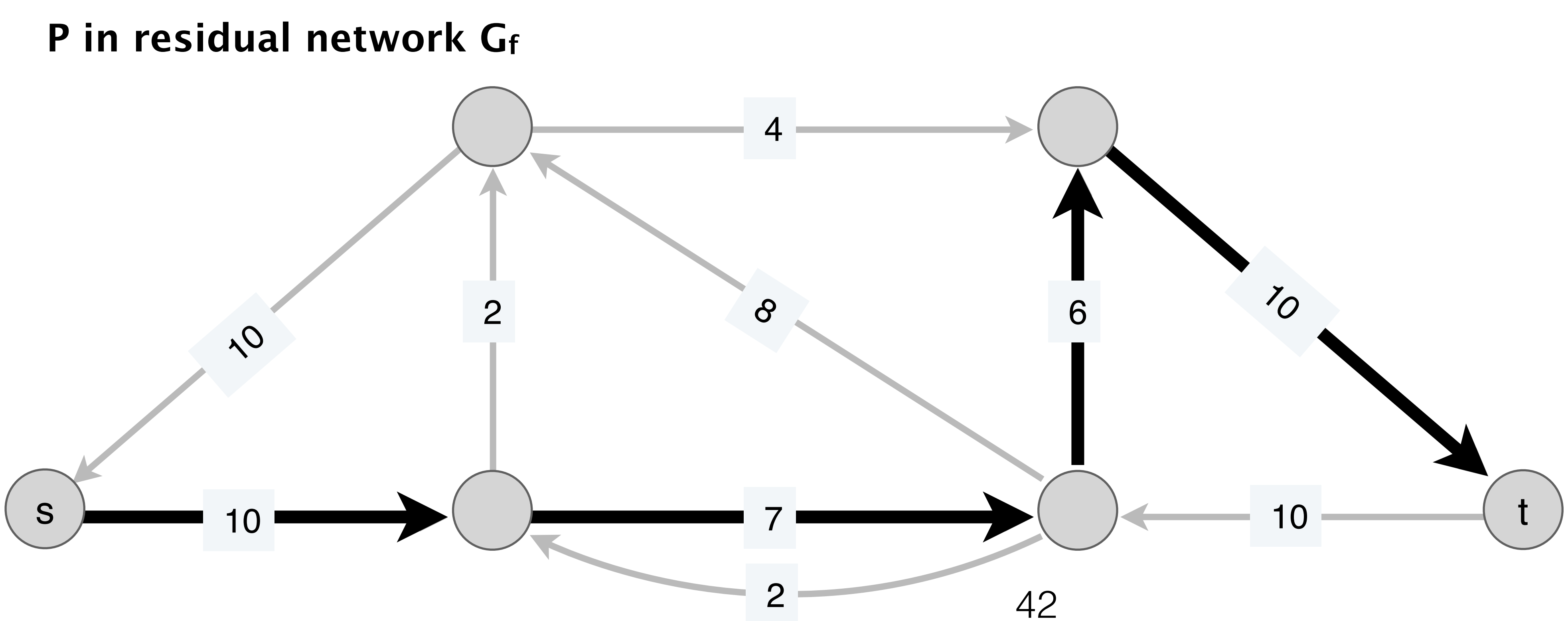


residual network G_f



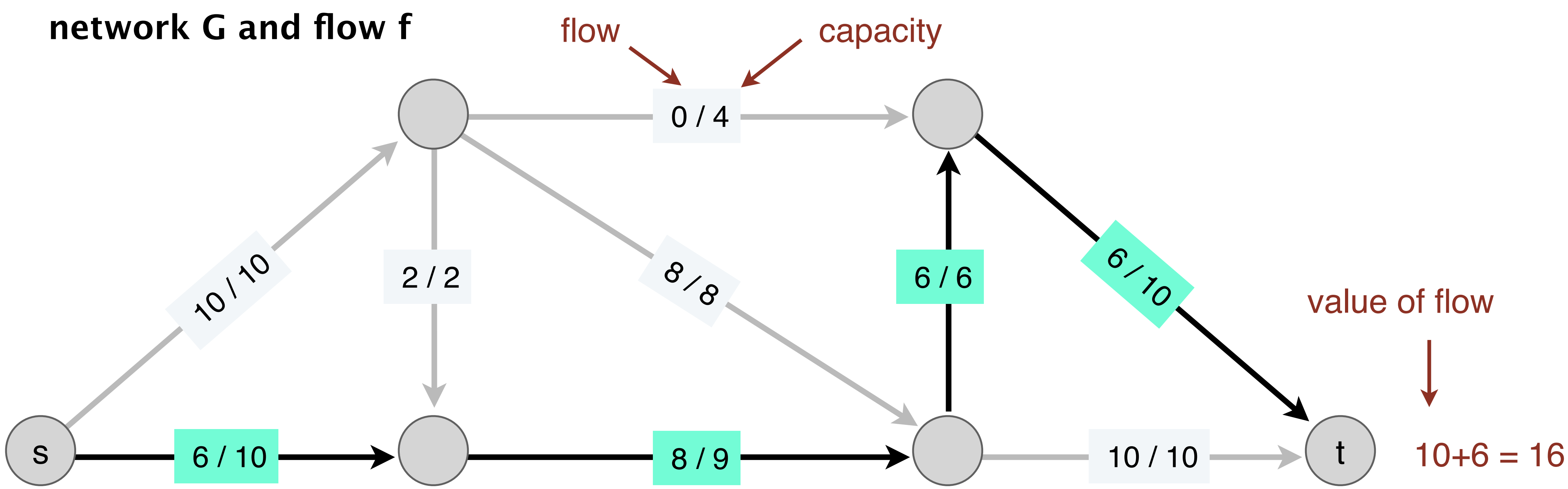
Ford-Fulkerson Example



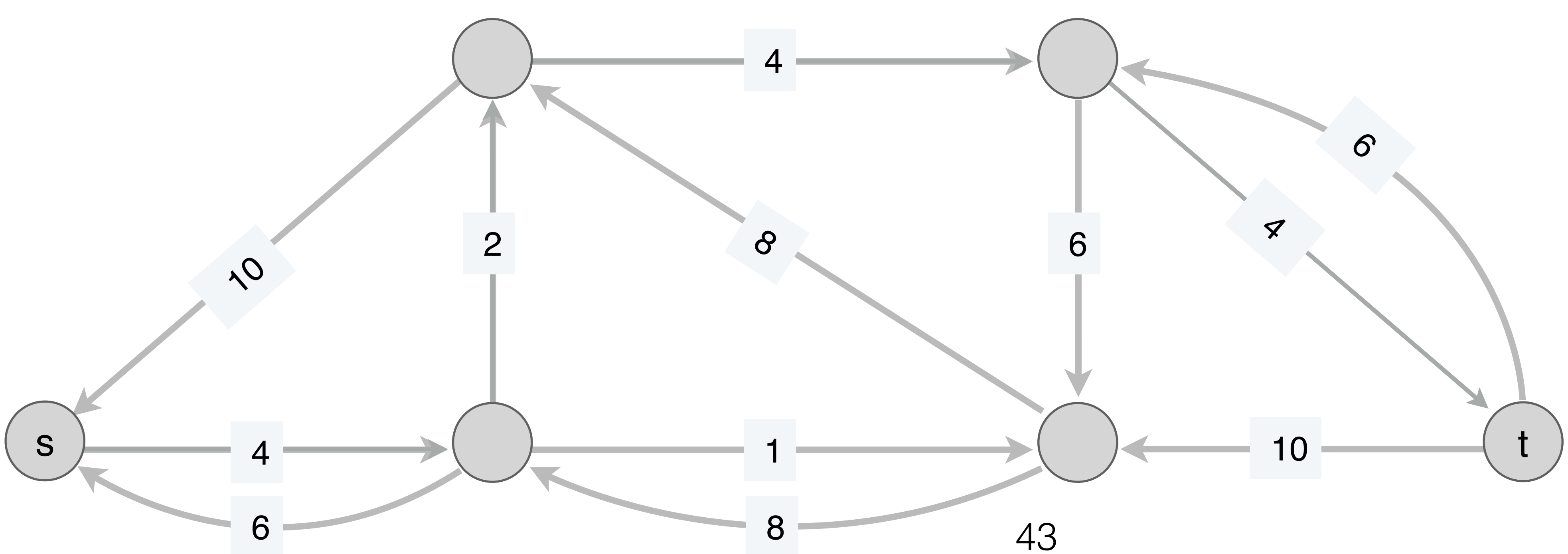


Ford-Fulkerson Example

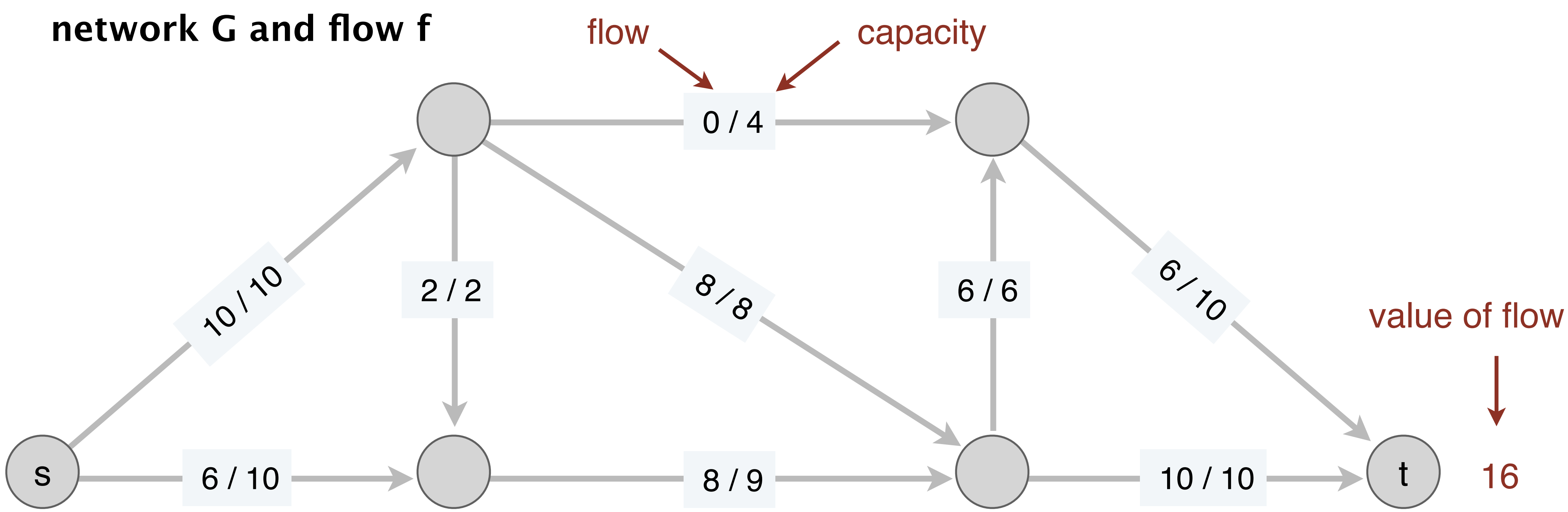
network G and flow f

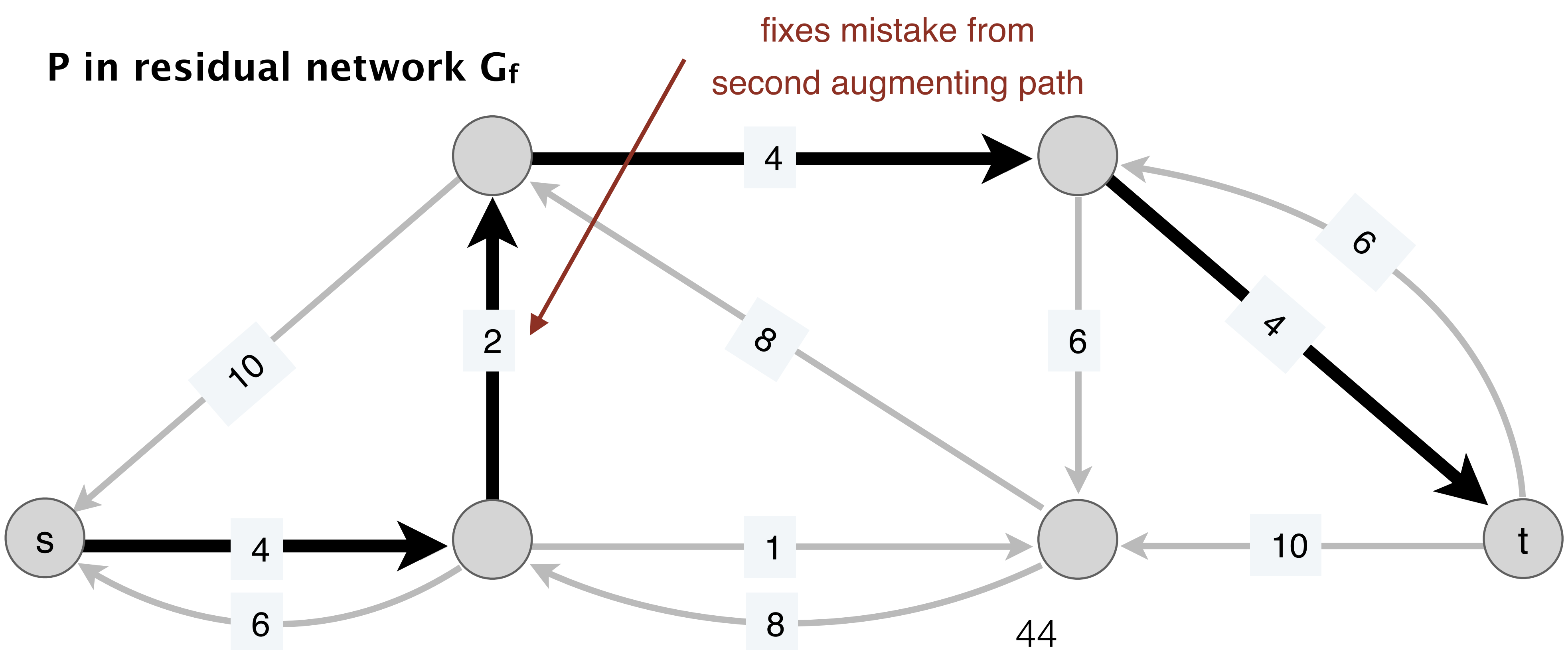


residual network G_f



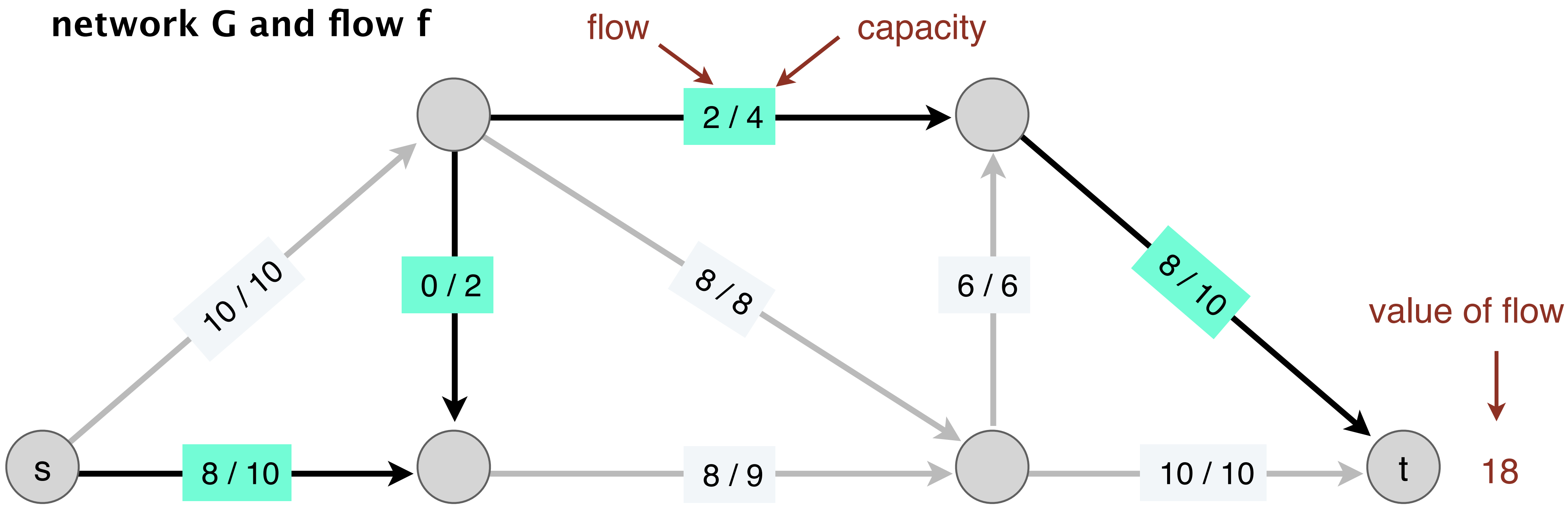
Ford-Fulkerson Example



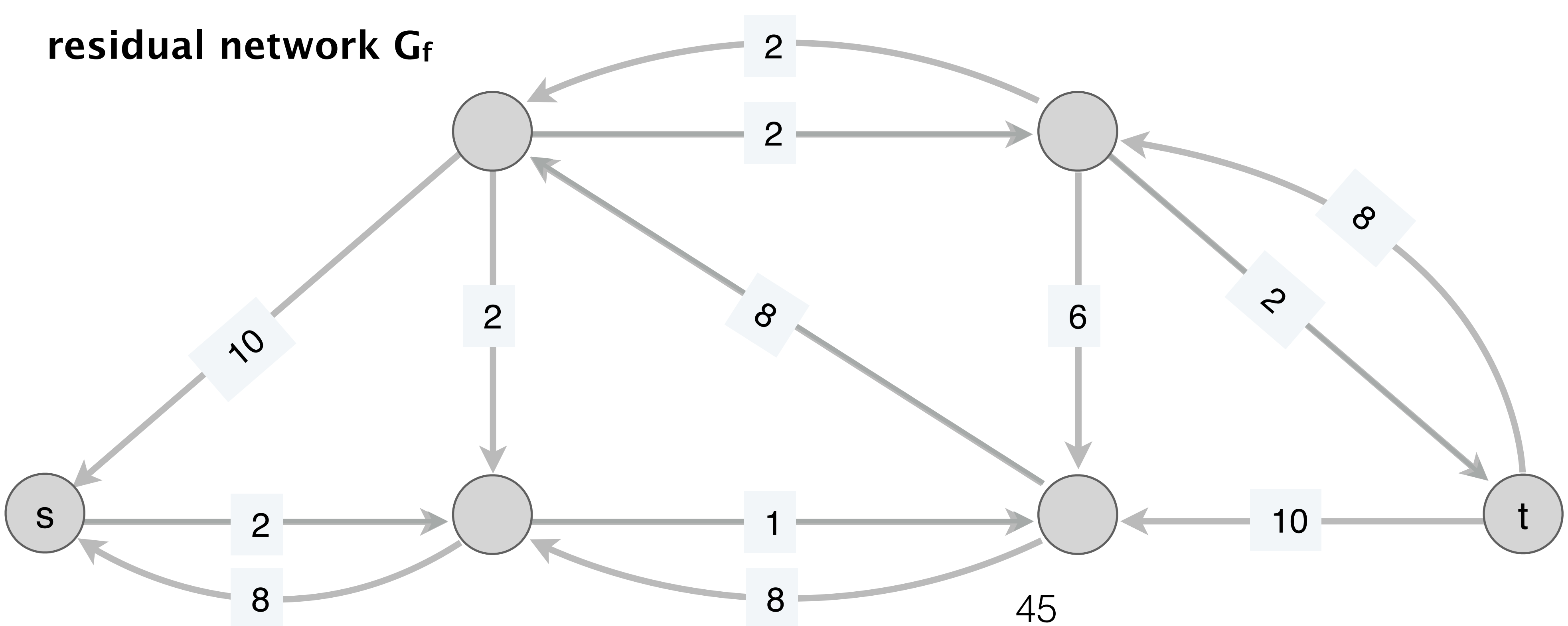


Ford-Fulkerson Example

network G and flow f

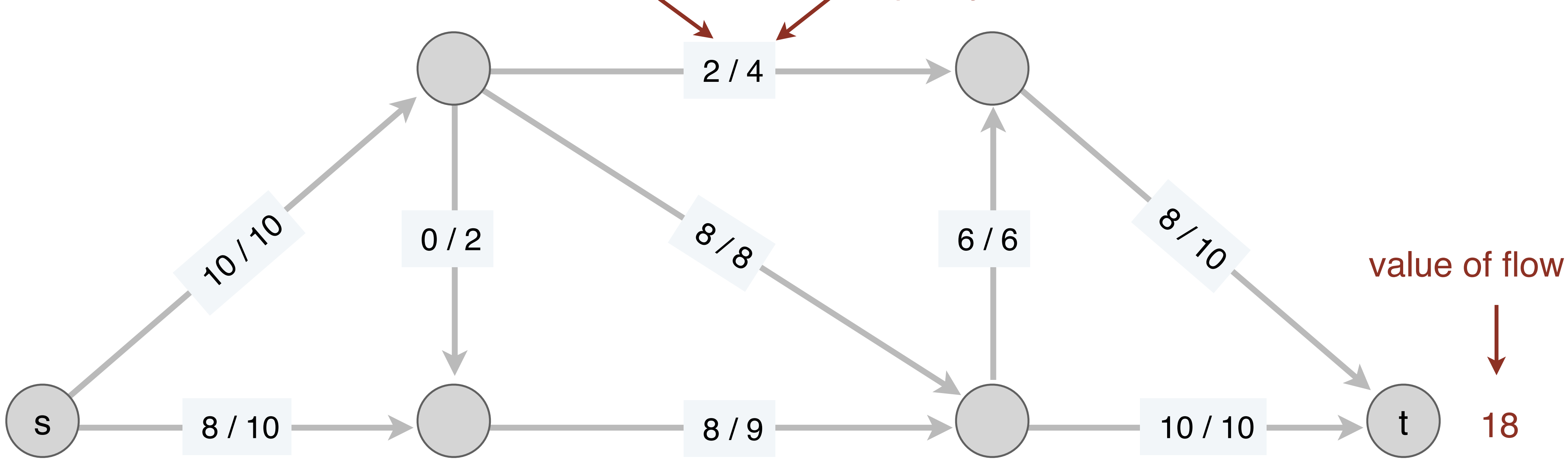


residual network G_f

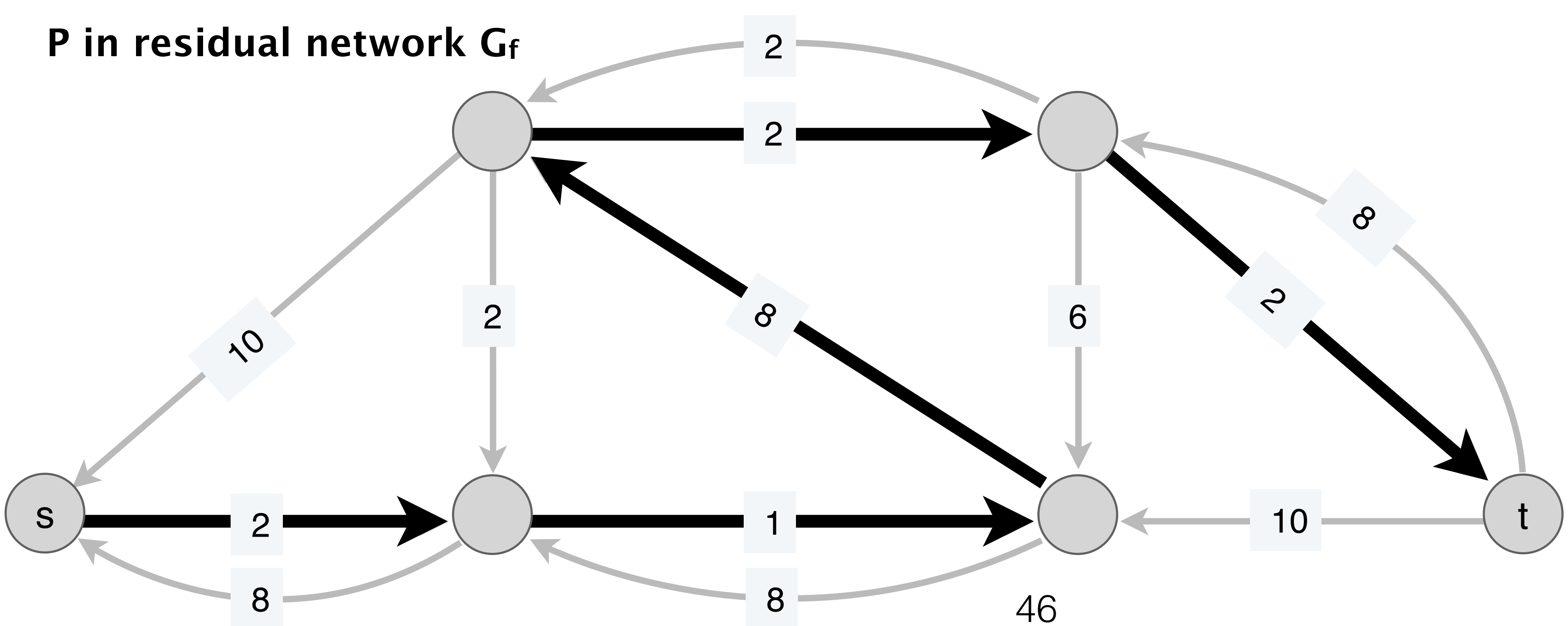


Ford-Fulkerson Example

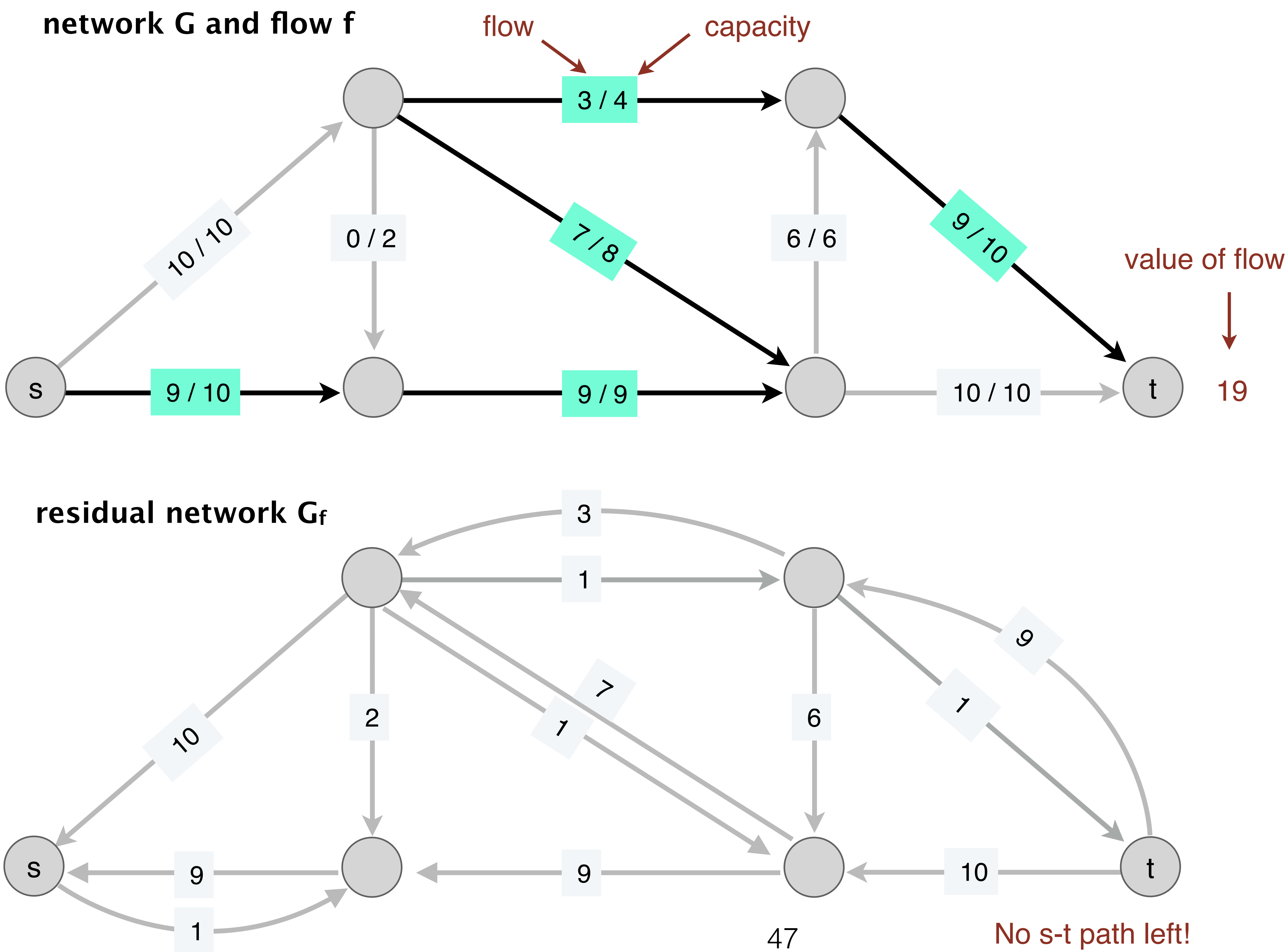
network G and flow f



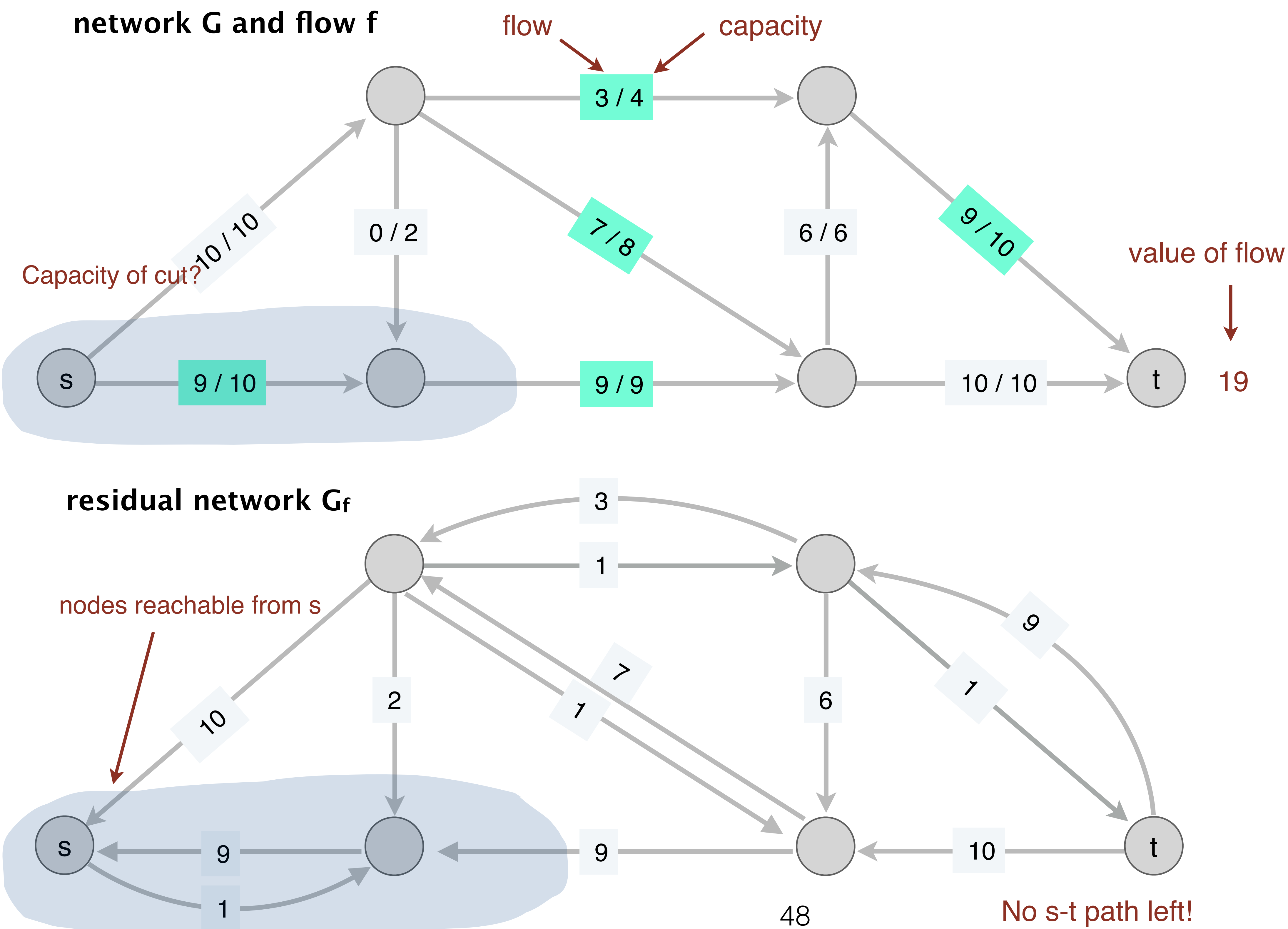
P in residual network G_f



Ford-Fulkerson Example



Ford-Fulkerson Example



Correctness & Value of Flow

Augmenting Path & Flow

- **Claim**. Let f be a feasible flow in G and let P be an augmenting path in G_f with bottleneck capacity b. Let $f' \leftarrow$ AUGMENT(f, P), then f' is **a feasible flow** and $v(f') = v(f) + b$.

- **Proof**. Only need to verify constraints on the edges of P (since $f' = f$ for other edges). Let $e = (u, v) \in P$

  - If e is a forward edge:
    $$\begin{aligned} f(e) &\le f'(e) \\ &\le f(e) + b \\ &\le f(e) + (c_e - f(e)) = c_e \end{aligned}$$

  - If e is a backward edge:

  - $$\begin{aligned} f(e) &\ge f'(e) = f(e) - b \\ &\ge f(e) - f(e) = 0 \end{aligned}$$

- Conservation constraint hold on nodes in P (exercise)

Augmenting Path & Flow

- **Claim**. Let f be a feasible flow in G and let P be an augmenting path in G_f with bottleneck capacity b. Let $f' \leftarrow$ AUGMENT(f, P), then f' is a feasible flow and $v(f') = v(f) + b$.
- **Proof**.
  - First edge $e \in P$ must be out of s in G_f
  - P is simple so never visits s again
  - e must be a forward edge (P is a path from s to t)
  - Thus $f(e)$ increases by b, increasing $v(f)$ by b

    ∎

Optimality

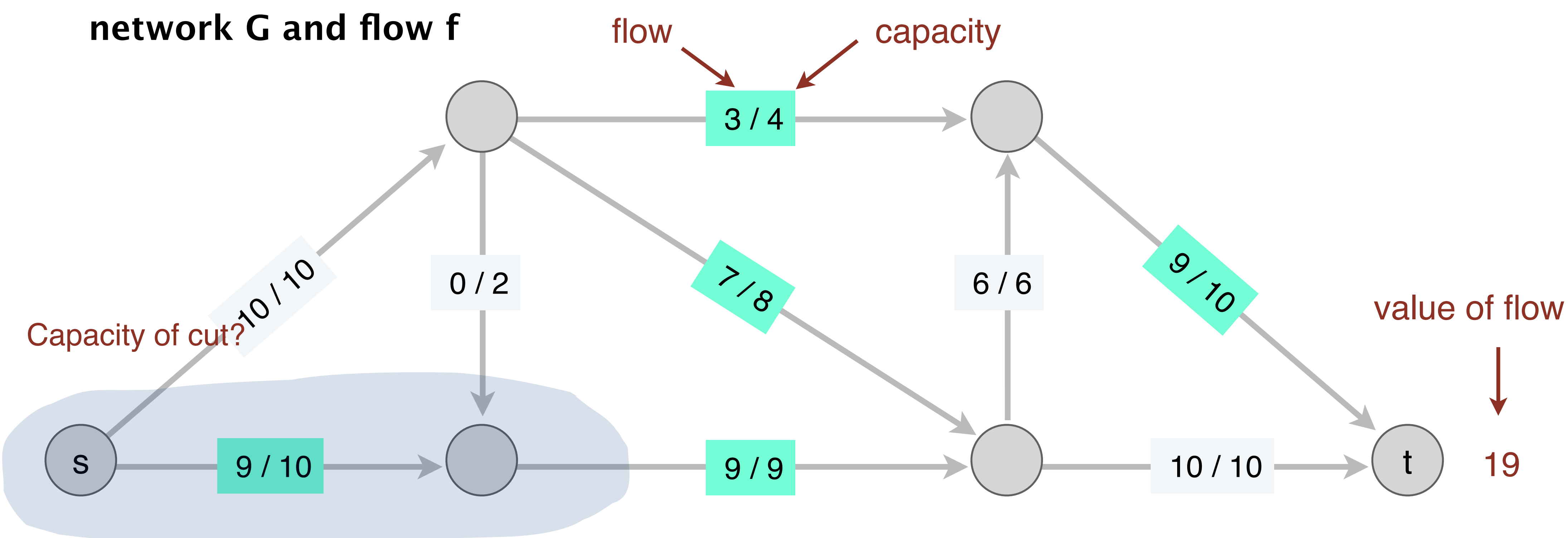
Ford-Fulkerson Optimality

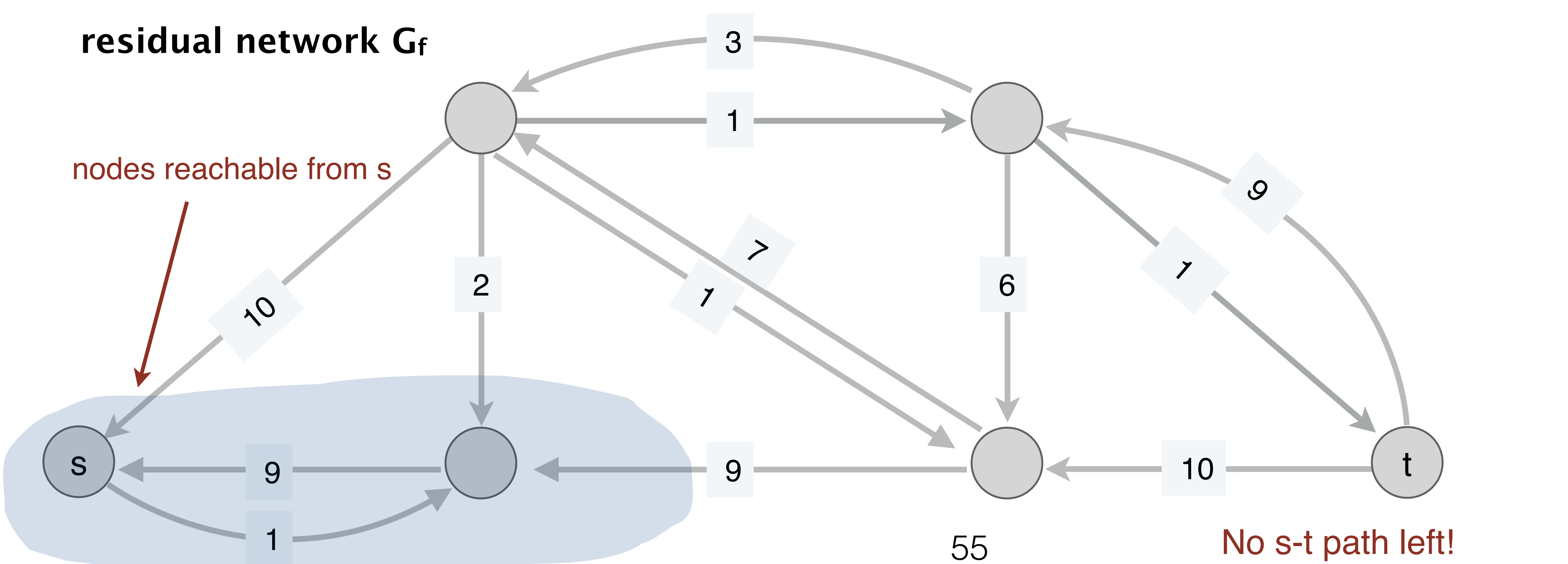
- **Recall**: If f is any feasible s-t flow and (S, T) is any s-t cut then $v(f) \leq c(S, T)$.
- We will show that the Ford-Fulkerson algorithm terminates in a flow that achieves equality, that is,
- Ford-Fulkerson finds a flow f^* and there exists a cut (S^*, T^*) such that
$$v(f^*) = c(S^*, T^*)$$
- Proving this shows that it finds the maximum flow!
- This also **proves the max-flow min-cut theorem**

Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**.
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$,
  $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \varnothing$
- Consider an edge $e = u \to v$ with $u \in S^*, v \in T^*$, then what can we say about $f(e)$?

Recall: Ford-Fulkerson Example





Ford-Fulkerson Optimality

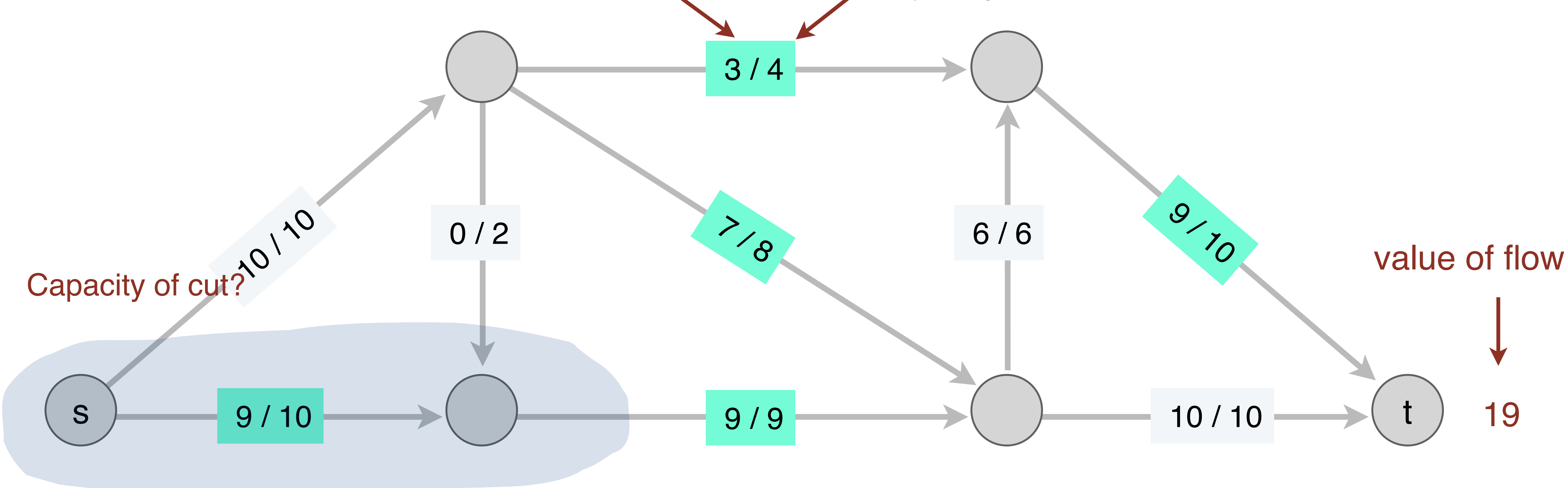
- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**.
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$,
$T^* = V - S^*$
- Is this an s-t cut?
    - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = u \to v$ with $u \in S^*, v \in T^*$,
then what can we say about $f(e)$?
    - $f(e) = c(e)$

Ford-Fulkerson Optimality

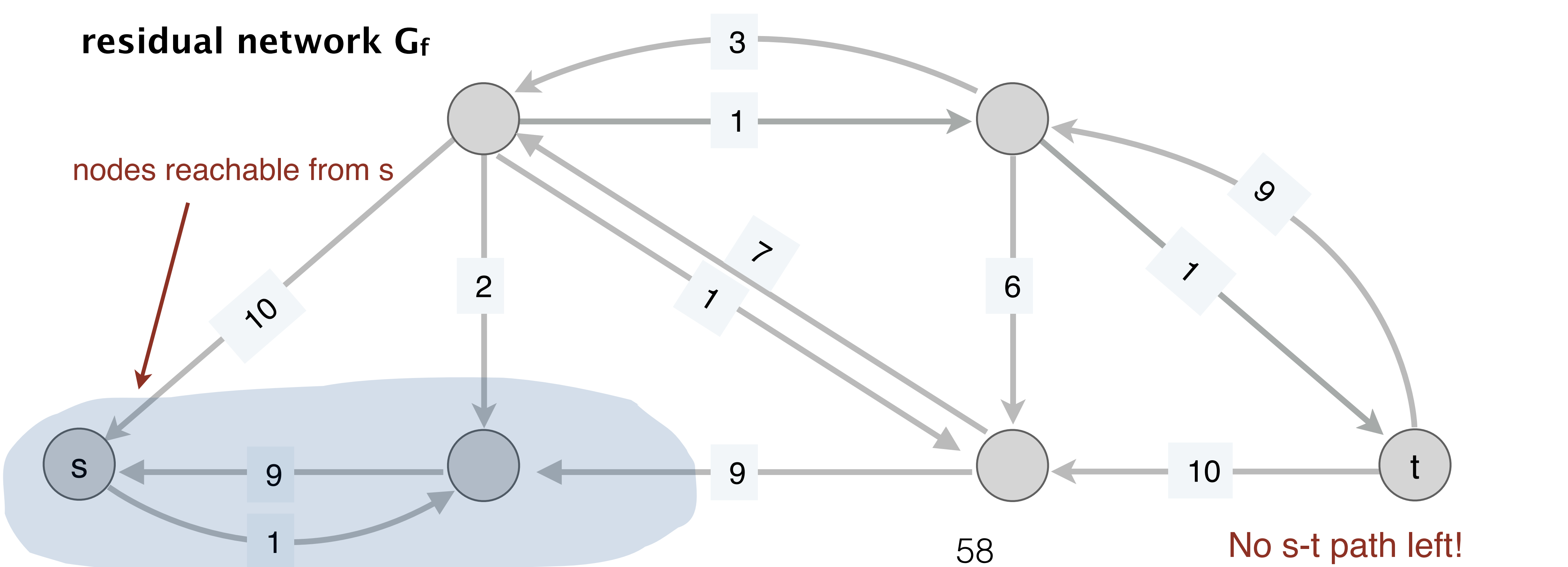
- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$,
  $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = w \to v$ with
  $v \in S^*, w \in T^*$, then what can we say about $f(e)$?

Recall: Ford-Fulkerson Example

network G and flow f



residual network G_f



Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v$ is reachable from s in $G_f\}$,
  $T^* = V - S^*$
- Is this an s-t cut?
  - $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \varnothing$
- Consider an edge $e = w \to v$ with
  $v \in S^*, w \in T^*$, then what can we say about $f(e)$?
  - $f(e) = 0$

Ford-Fulkerson Optimality

- **Lemma**. Let f be a s-t flow in G such that there is no augmenting path in the residual graph G_f, then there exists a cut (S^*, T^*) such that $v(f) = c(S^*, T^*)$.
- **Proof**. **(Cont.)**
- Let $S^* = \{v \mid v \text{ is reachable from } s \text{ in } G_f\}$, $T^* = V - S^*$
- Thus, all edges leaving S^* are completely saturated and all edges entering S^* have zero flow
- $v(f) = f_{out}(S^*) - f_{in}(S^*) = f_{out}(S^*) = c(S^*, T^*)$ ∎
- **Corollary**. Ford-Fulkerson returns the maximum flow.

Ford-Fulkerson Algorithm
Running Time

Ford-Fulkerson Performance

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s↝t path P in G_f)

  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

RETURN f.

- Does the algorithm terminate?
- Can we bound the number of iterations it does?
- Running time?

Ford-Fulkerson Running Time

- Recall we proved that with each call to AUGMENT, we increase **value of flow** by $b = \text{bottleneck}(G_f, P)$
- **Assumption**. Suppose all capacities $c(e)$ are integers.
- **Integrality invariant.** Throughout Ford–Fulkerson, every edge flow $f(e)$ and corresponding residual capacity is an integer. Thus $b \geq 1$.
- Let $C = \max_u c(s \to u)$ be the maximum capacity among edges leaving the source s.
- It must be that $v(f) \leq (n-1)C = O(nC)$
- Since, $v(f)$ increases by $b \geq 1$ in each iteration, it follows that FF algorithm terminates in at most $v(f) = O(nC)$ iterations.

Ford-Fulkerson Running Time

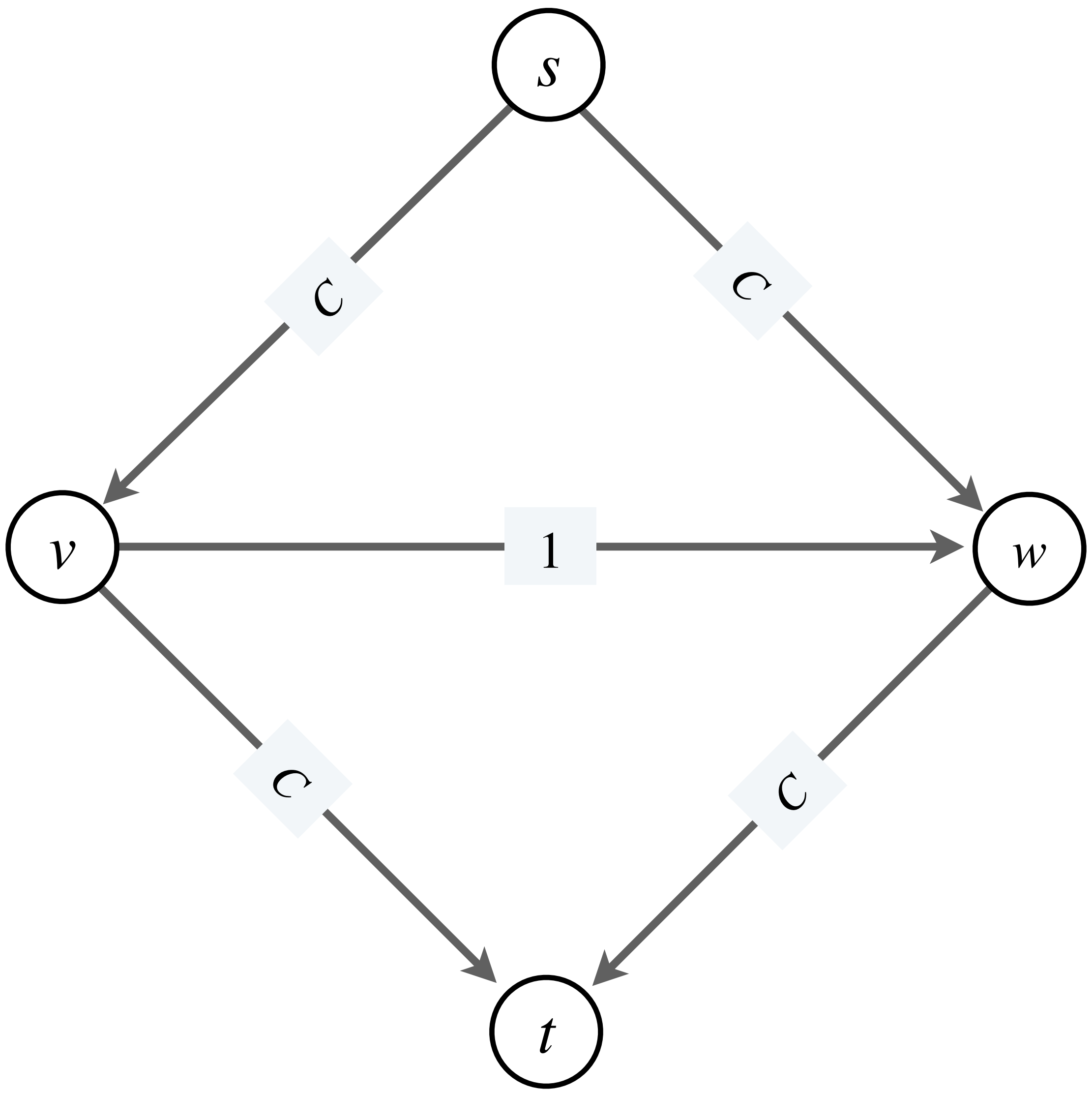
- **Claim.** Ford-Fulkerson can be implemented to run in time $O(nmC)$, where $m = |E| \geq n - 1$ and $C = \max_u c(s \to u)$.
- **Proof**. We know algorithm terminates in at most C iterations. Each iteration takes $O(m)$ time:
  - We need to find an augmenting path in G_f
  - G_f has at most $2m$ edges, using BFS/DFS takes $O(m + n) = O(m)$ time
  - Augmenting flow in P takes $O(n)$ time
  - Given new flow, we can build new residual graph in $O(m)$ time ∎

[Digging Deeper] Polynomial time?

- Does the Ford-Fulkerson algorithm run in time polynomial in the input size?
- Running time is $O(nmC)$, where $C = \max_u c(s \to u)$, suppose it is even larger, that is, $C = \max_e c(e)$
- What is the input size?
- Let's take an example

[Digging Deeper] Polynomial time?

- **Question**. Does the Ford-Fulkerson algorithm run in polynomial-time in the size of the input? ← ~ m, n, and $\log C$
- **Answer**. No. if max capacity is C, the algorithm can take $\geq C$ iterations. Consider the following example.



- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- …
- $s \to v \to w \to t$
- $s \to w \to v \to t$

each augmenting path sends only 1 unit of flow (# augmenting paths = $2C$)

[Digger Deeper] Pseudo-Polynomial

- Input graph has n nodes and $m = O(n^2)$ edges, each with capacity c_e
- $C = \max_{e \in E} c(e)$, then $c(e)$ takes $O(\log C)$ bits to represent
- Input size: $O(n \log n + m \log n + m \log C)$ bits
- Let $t = \log n$, $b = \log C$
- Input size: $O(nv + m(v + b))$
- Running time: $O(nm2^b)$, exponential in the size of C
- Such algorithms are called **pseudo-polynomial**
    - If the running time is polynomial in the **magnitude** but **not size** of an input parameter.

Non-Integral Capacities?

- If the capacities are rational, can just multiply to obtain a large integer (massively increases running time)
- If capacities are irrational, Ford-Fulkerson can run infinitely!
    - Idea: amount of flow sent decreases by a constant factor each loop

Summary

- Given a flow network with integer capacities, Ford-Fulkerson computes the **max flow** in $O(mnC)$ time
- A **constructive proof** of the max-flow min-cut theorem
- It is a pseudo-polynomial algorithm
    - Can take exponential time wrt to size of C
    - Bad performance in the worst case can be blamed on poor augmenting path choices
- **Next. (Flow Applications)** Solving other optimization problems by reduction them to a network flow problem

Network Flow [Optional]:
Beyond Ford Fulkerson

Edmond and Karp's Algorithms

- Ford and Fulkerson's algorithm does not specify which path in the residual graph to augment
- Poor worst-case behavior of the algorithm can be blamed on bad choices on augmenting path
- **Better choice of augmenting paths.** In 1970s, Jack Edmonds and Richard Karp published two natural rules for choosing augmenting paths
  - Fattest augmenting paths first
  - Shortest (in terms of edges) augmenting paths first (Dinitz independently discovered & analyzed this rule)

Fattest Augmenting Paths First

- Ford Fulkerson is essentially a greedy algorithm way of augmenting paths:
    - Choose the augmenting path with largest bottleneck capacity
- Largest bottleneck path can be computed in $O(m \log n)$ time in a directed graph
    - Similar to Dijkstra's analysis
- How many iterations if we use this rule?
    - Won't prove this: takes $O(m \log C)$ iterations
- Overall running time is $O(m^2 \log n \log C)$ (polynomial time!)

Shortest Augmenting Paths First

- Choose the augmenting path with the smallest # of edges
- Can be found using BFS on G_f in $O(m + n) = O(m)$ time
- Surprisingly, this resulting a polynomial-time algorithm independent of the actual edge capacities !
- Analysis looks at "level" of vertices in the BFS tree of G_f rooted at s —levels only grow over time
- Analyzes # of times an edge $u \to v$ disappears from G_f
- Takes $O(mn)$ iterations overall
- Thus overall running time is $O(m^2 n)$

Progress on Network Flows

1951	$O(m\, n^2\, C)$	Dantzig
1955	$O(m\, n\, C)$	Ford–Fulkerson
1970	$O(m\, n^2)$	Edmonds–Karp, Dinitz
1974	$O(n^3)$	Karzanov
1983	$O(m\, n \log n)$	Sleator–Tarjan
1985	$O(m\, n \log C)$	Gabow
1988	$O(m\, n \log (n^2 / m))$	Goldberg–Tarjan
1998	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	$O(m\, n)$	Orlin
2014	$\tilde{O}(m\, n^{1/2} \log C)$	Lee–Sidford
2016	$\tilde{O}(m^{10/7}\, C^{1/7})$	Mądry

For unit capacity networks

Progress on Network Flows

- Best known: $O(nm)$
- Best lower bound?
  - None known. (Needs $\Omega(n+m)$ just to look at the network, but that's it)
- Some of these algorithms do REALLY well in "practice;" basically $O(n+m)$
- Well-known open problem

Summary

- Given a flow network with integer capacities, the **maximum flow and minimum cut** can be computed in $O(mn)$ time.
- **Next.** Network flow applications!

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)