Flow Networks: Max Flow

Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for each edge $e \in E$
- Find a simple $s \rightsquigarrow t$ path P in the residual network G_f
- Augment flow along path P by bottleneck capacity b
- 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 $s \rightsquigarrow t$ path P in G_f)

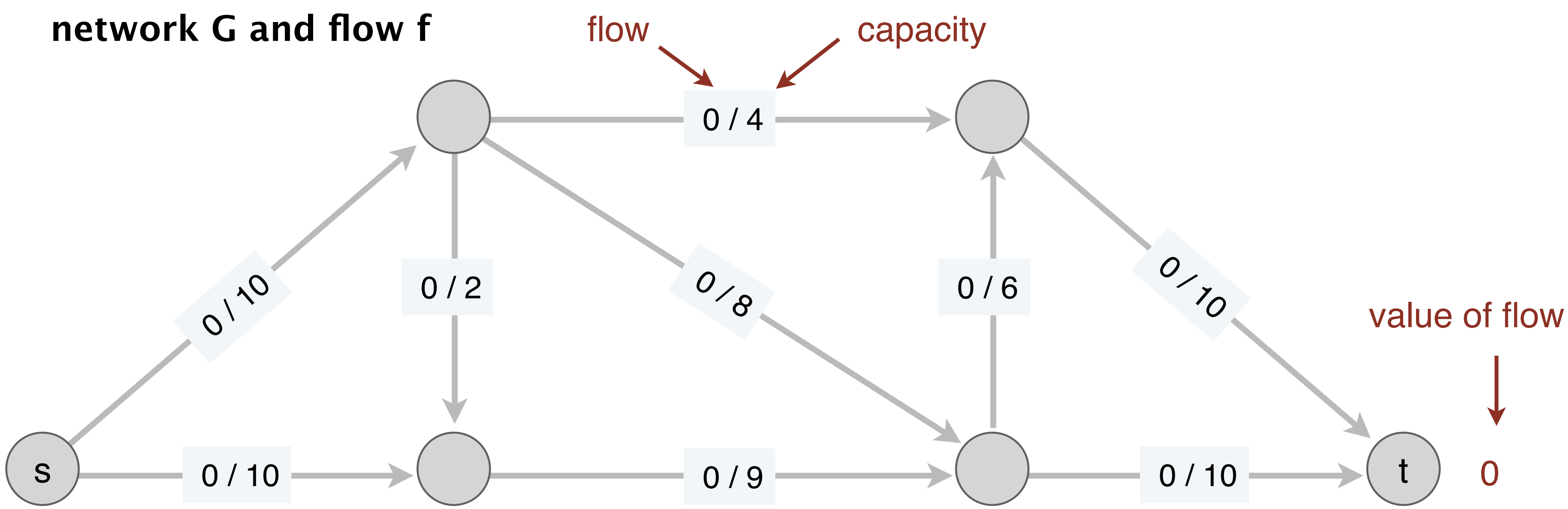
  $f \leftarrow$ AUGMENT(f, P).

  Update G_f.

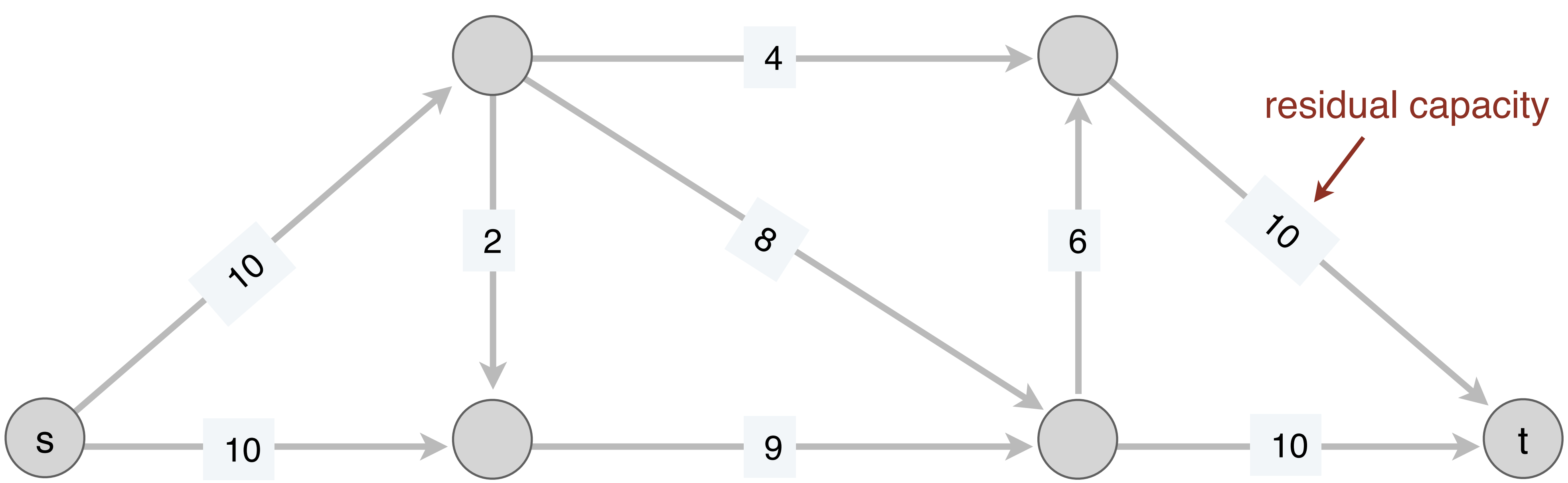
RETURN f.

Ford-Fulkerson Example

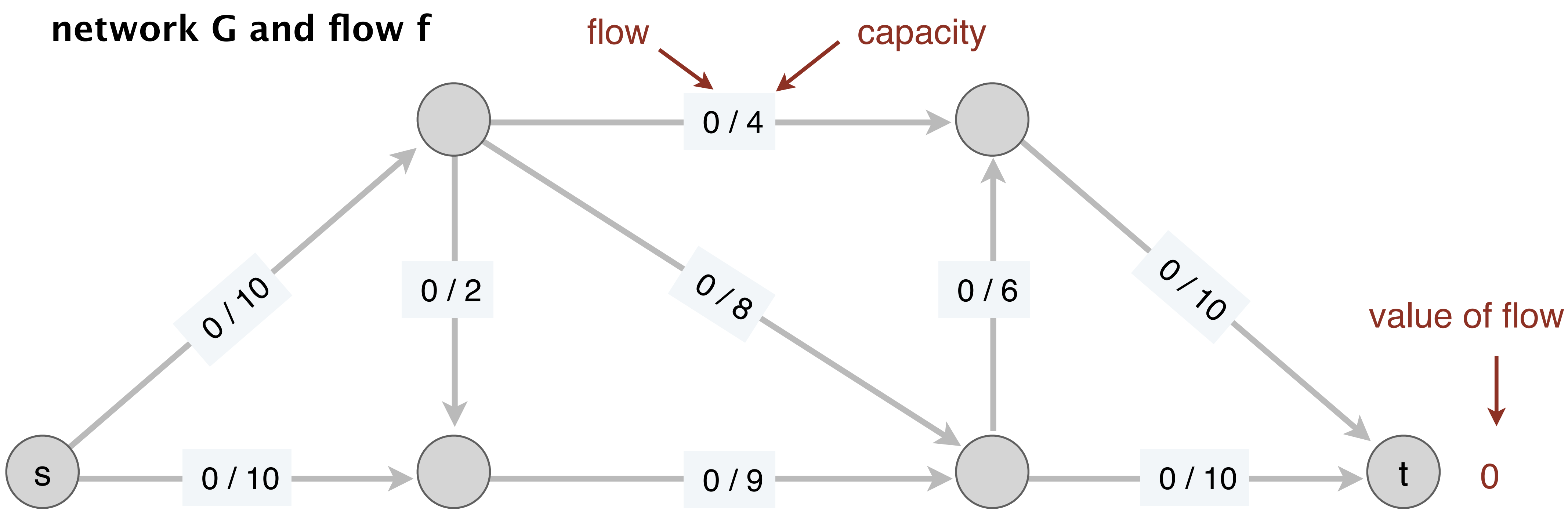
network G and flow f

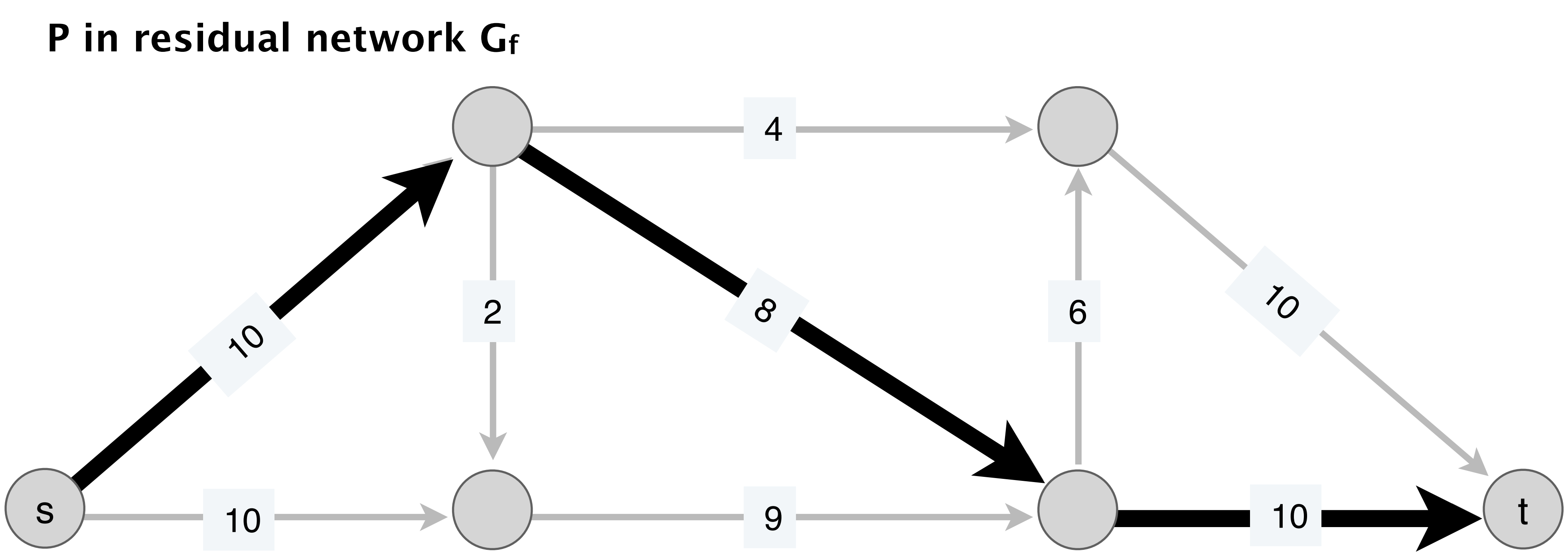


residual network G_f



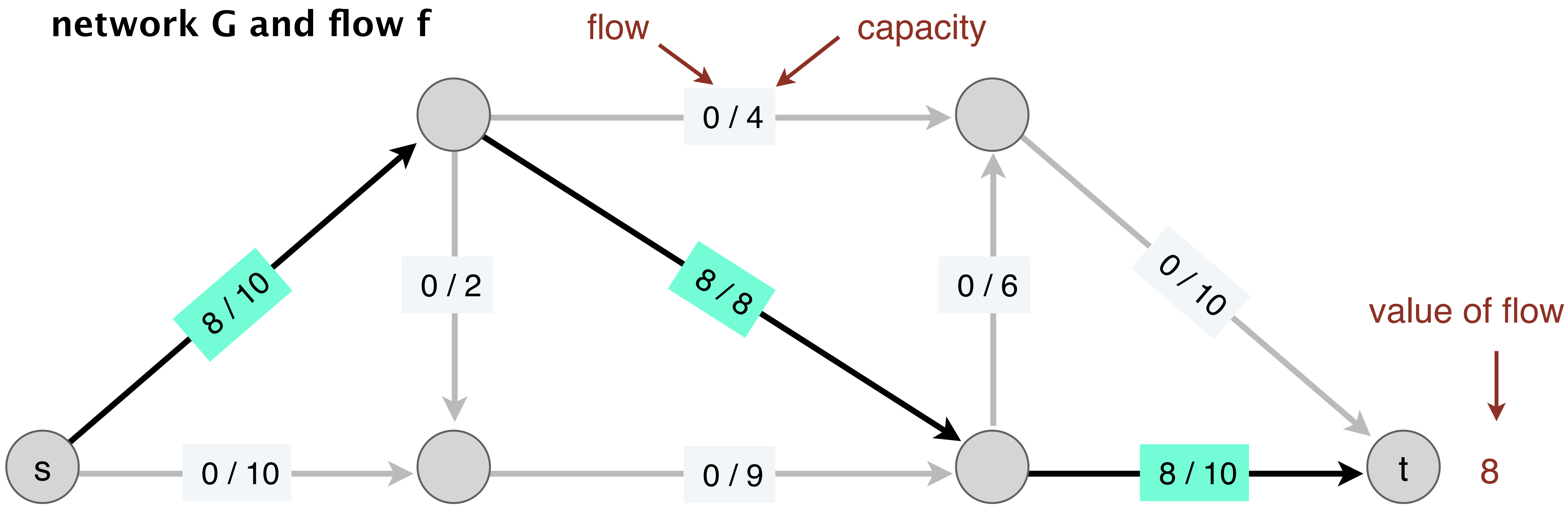
Ford-Fulkerson Example



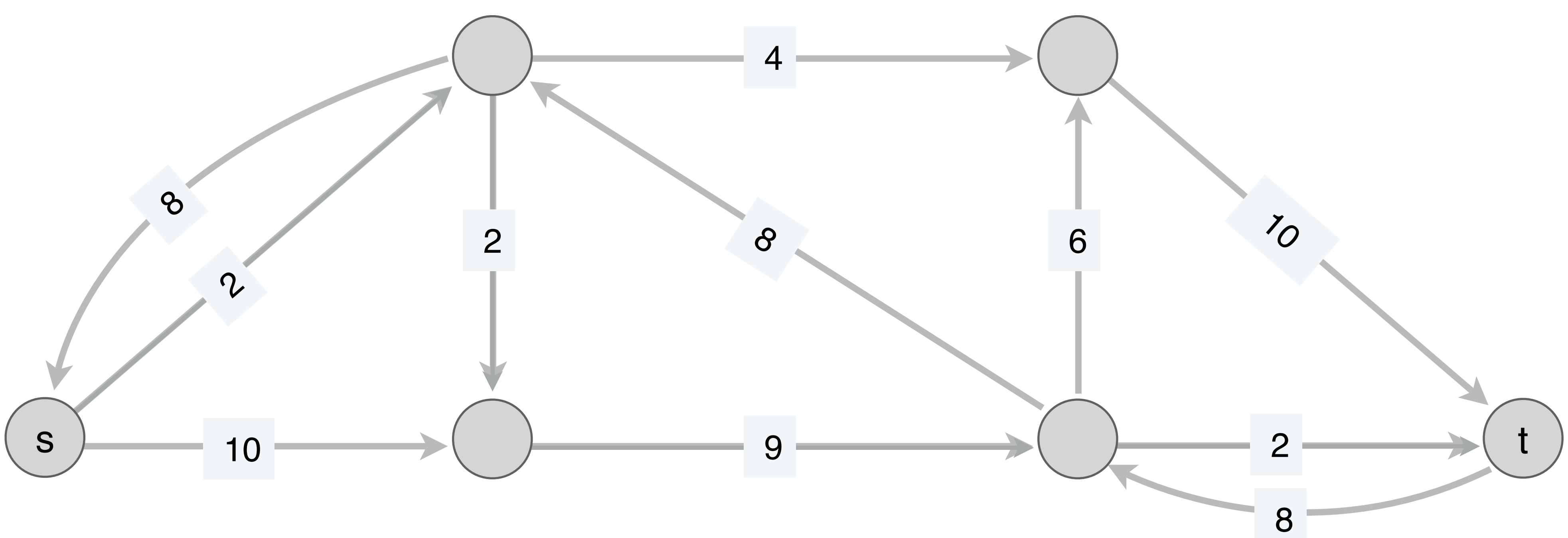


Ford-Fulkerson Example

network G and flow f

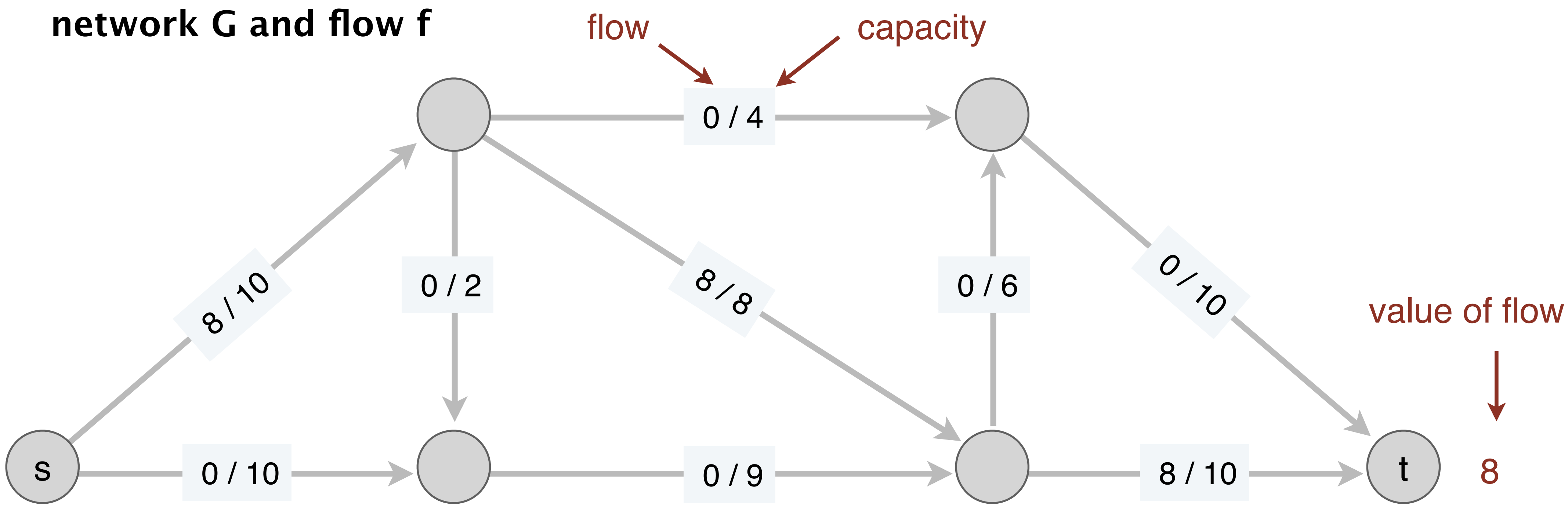


residual network G_f

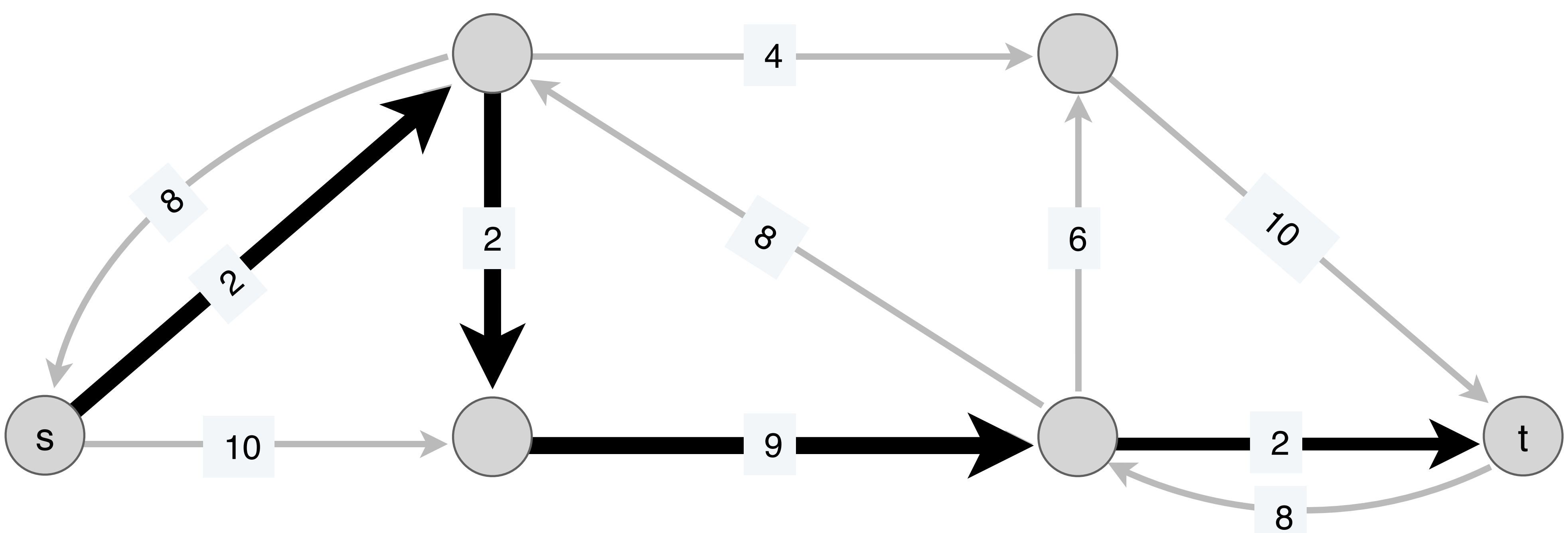


Ford-Fulkerson Example

network G and flow f

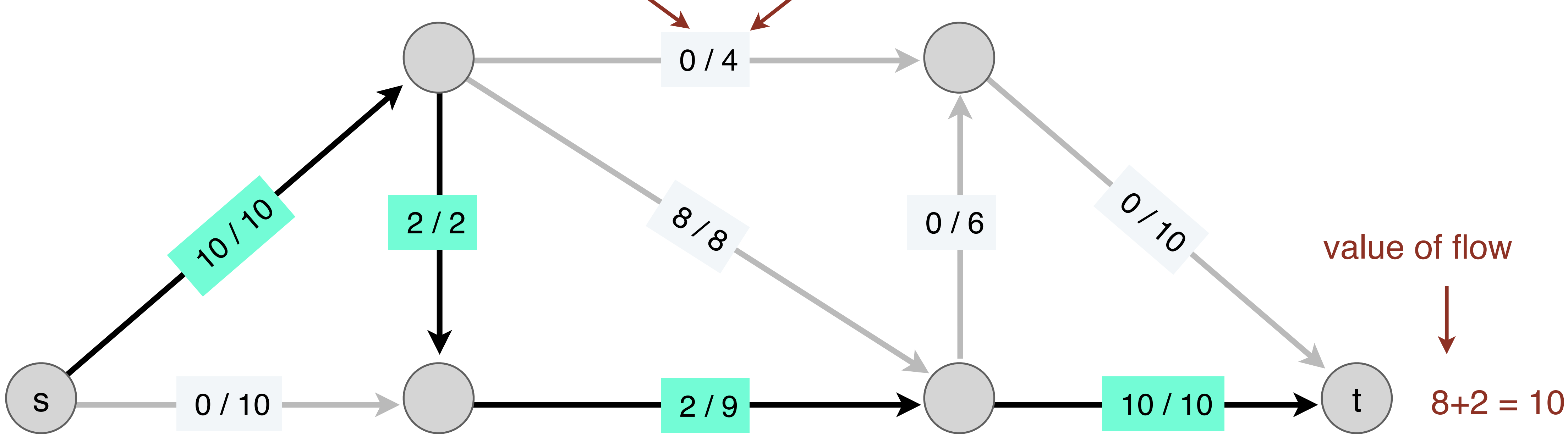


P in residual network G_f

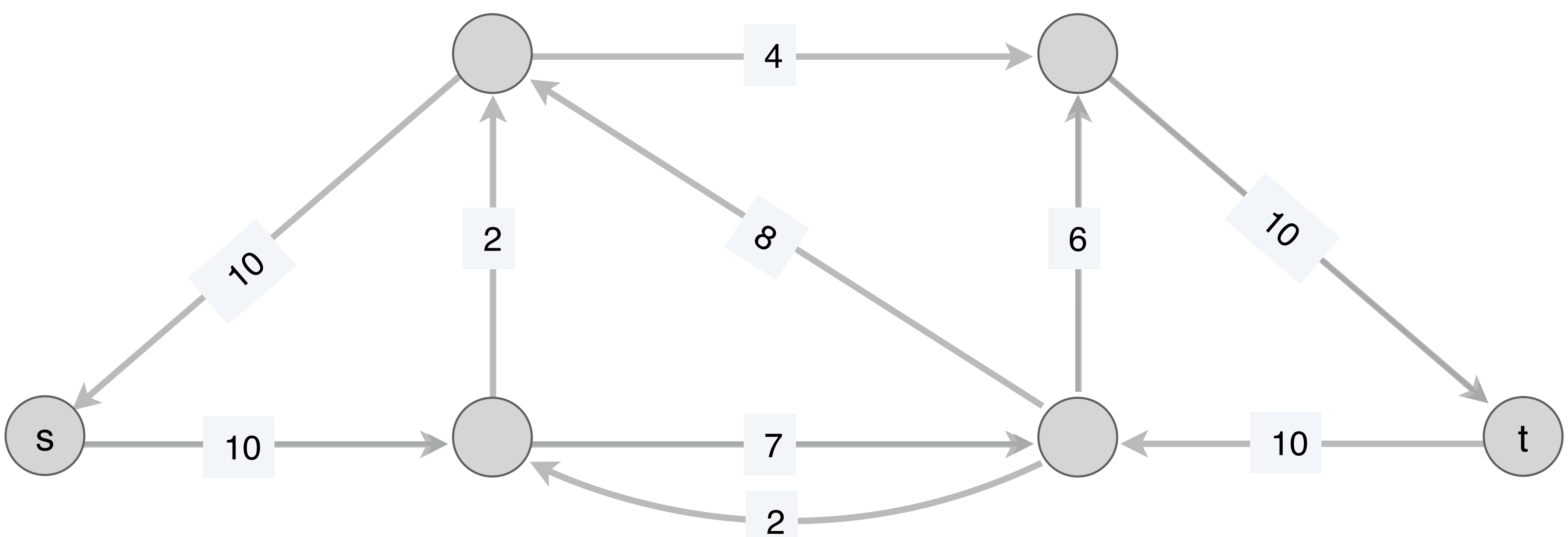


Ford-Fulkerson Example

network G and flow f

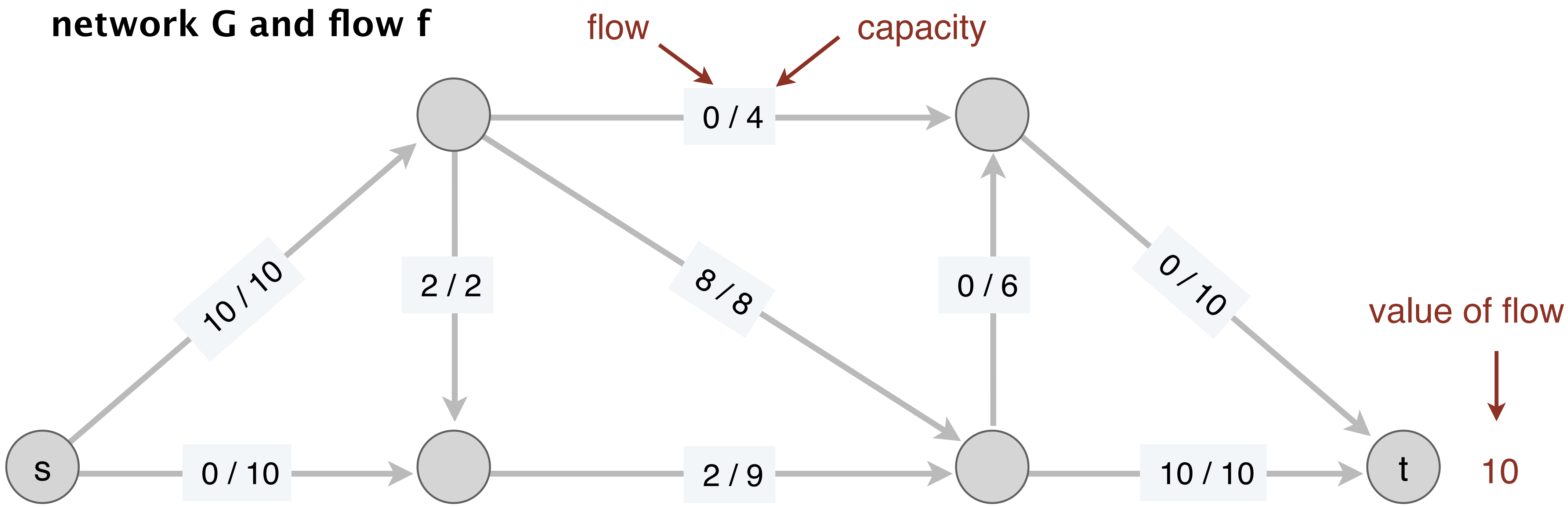


residual network G_f

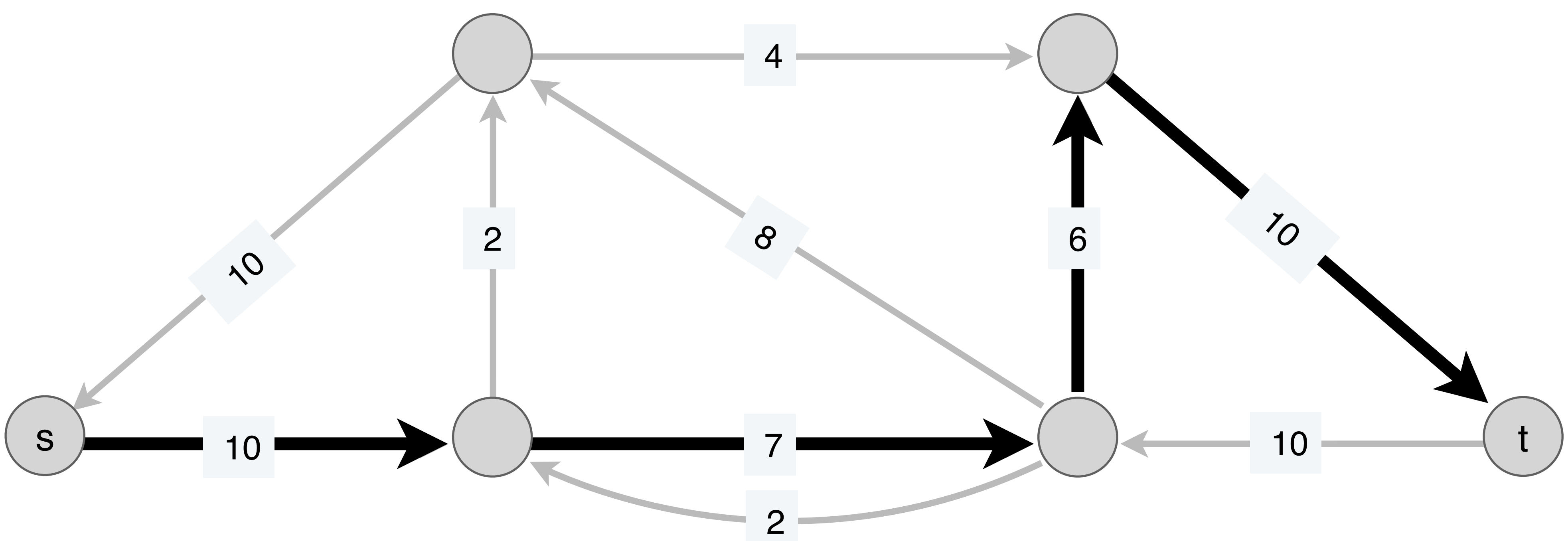


Ford-Fulkerson Example

network G and flow f

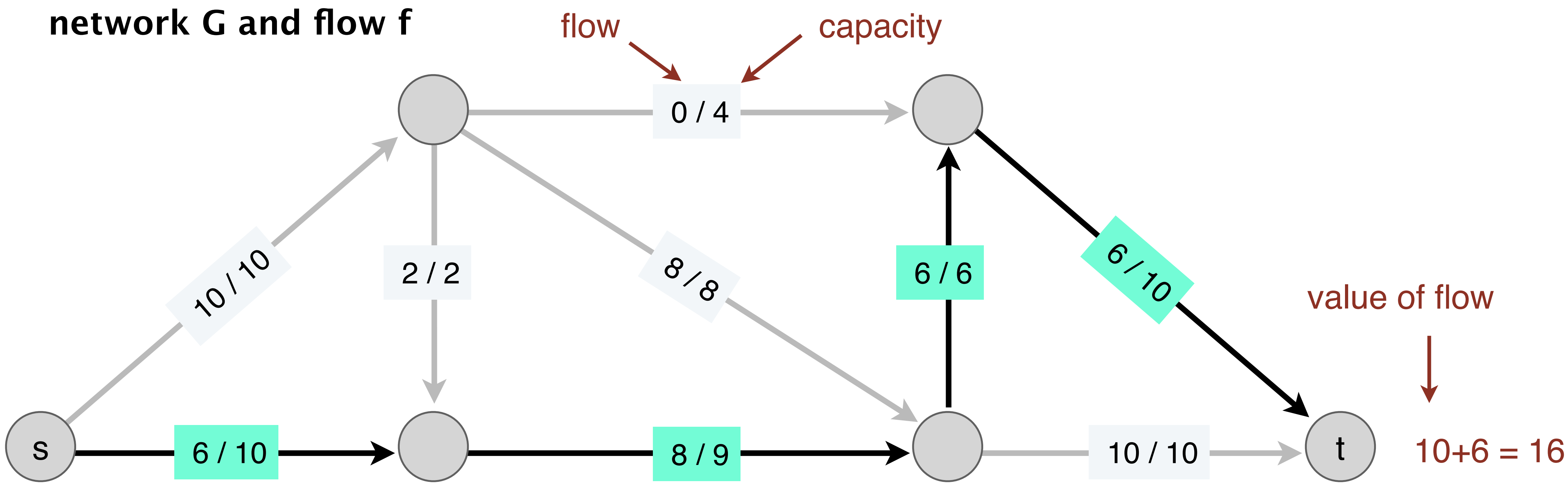


P in residual network G_f

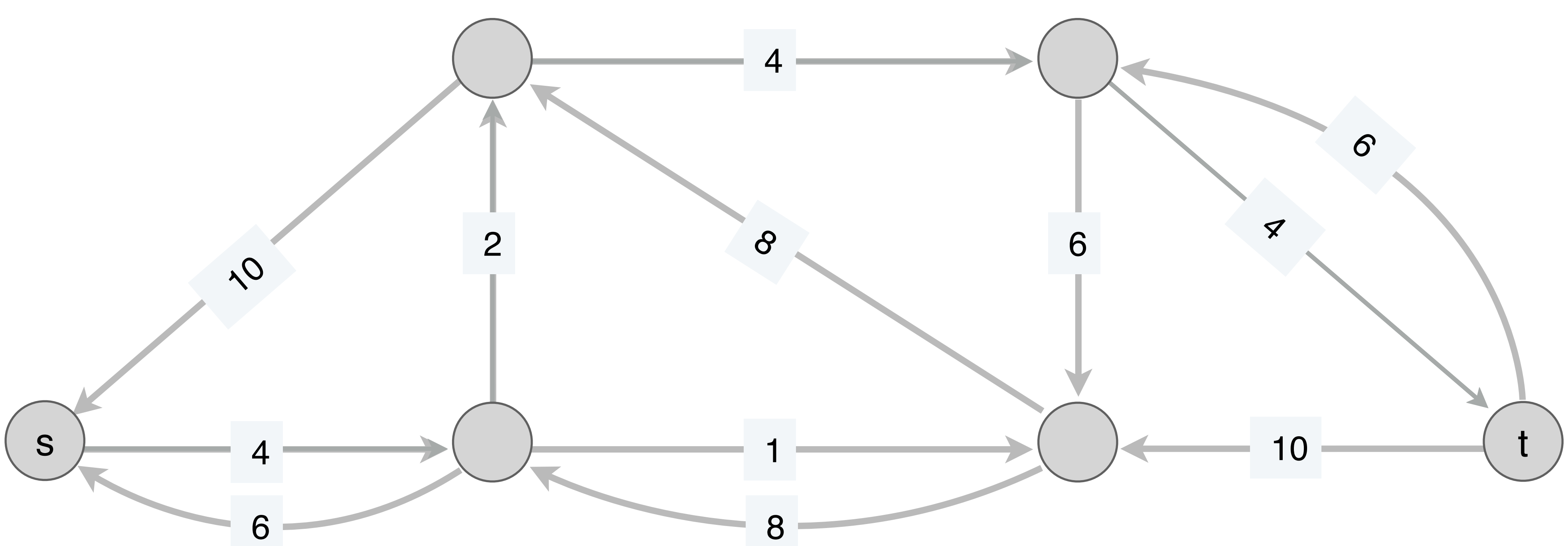


Ford-Fulkerson Example

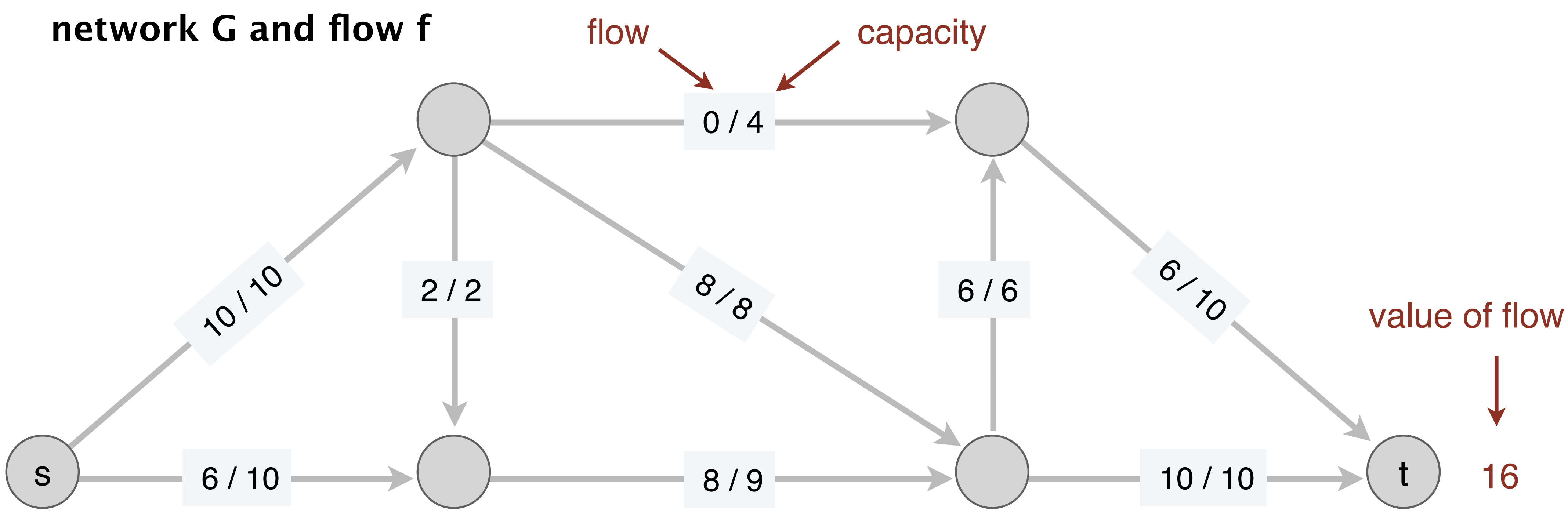
network G and flow f

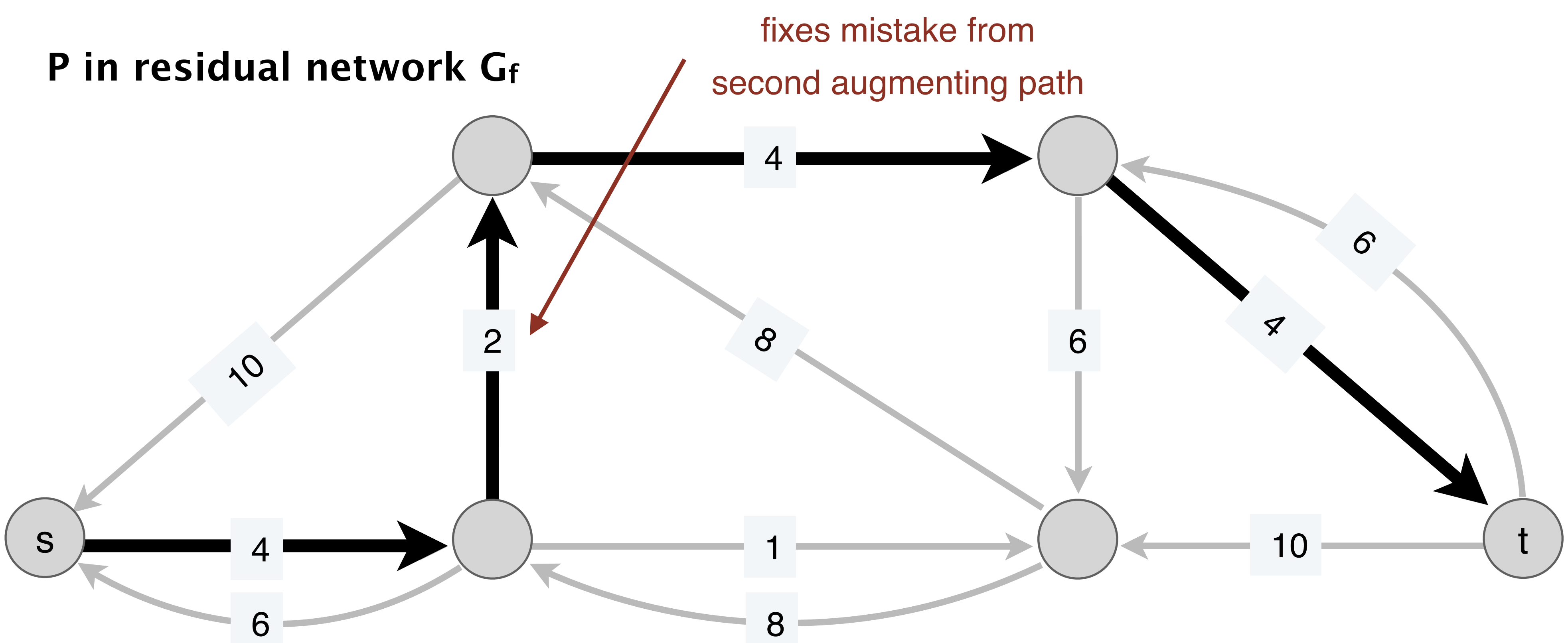


residual network G_f

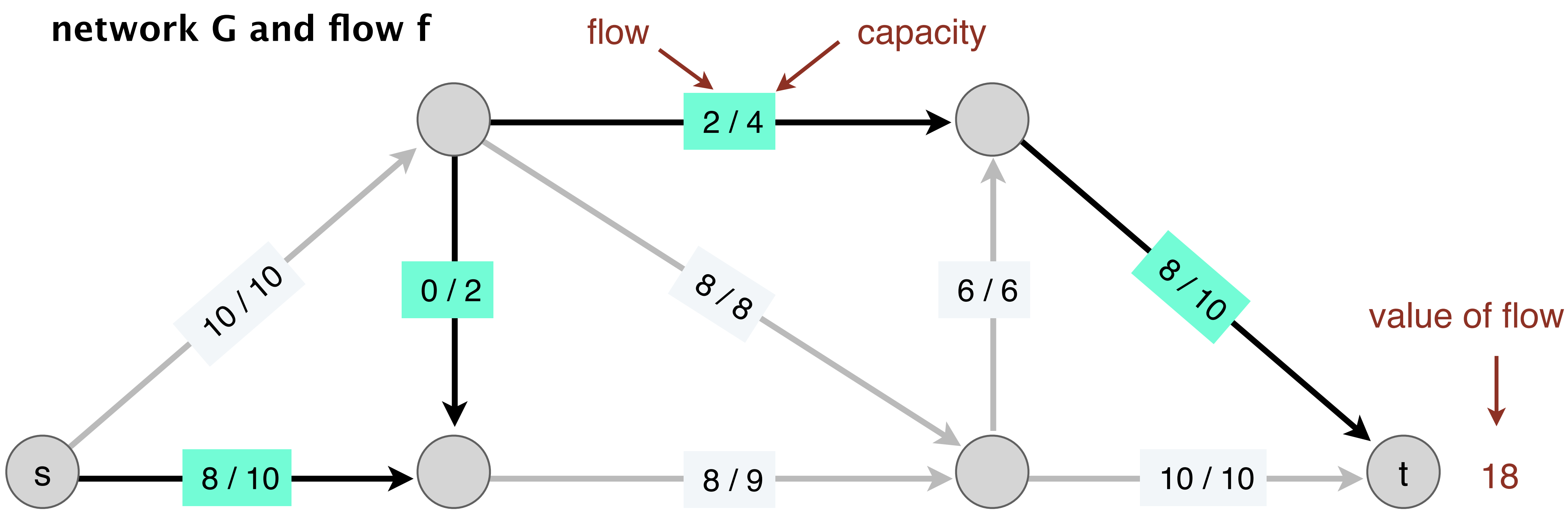


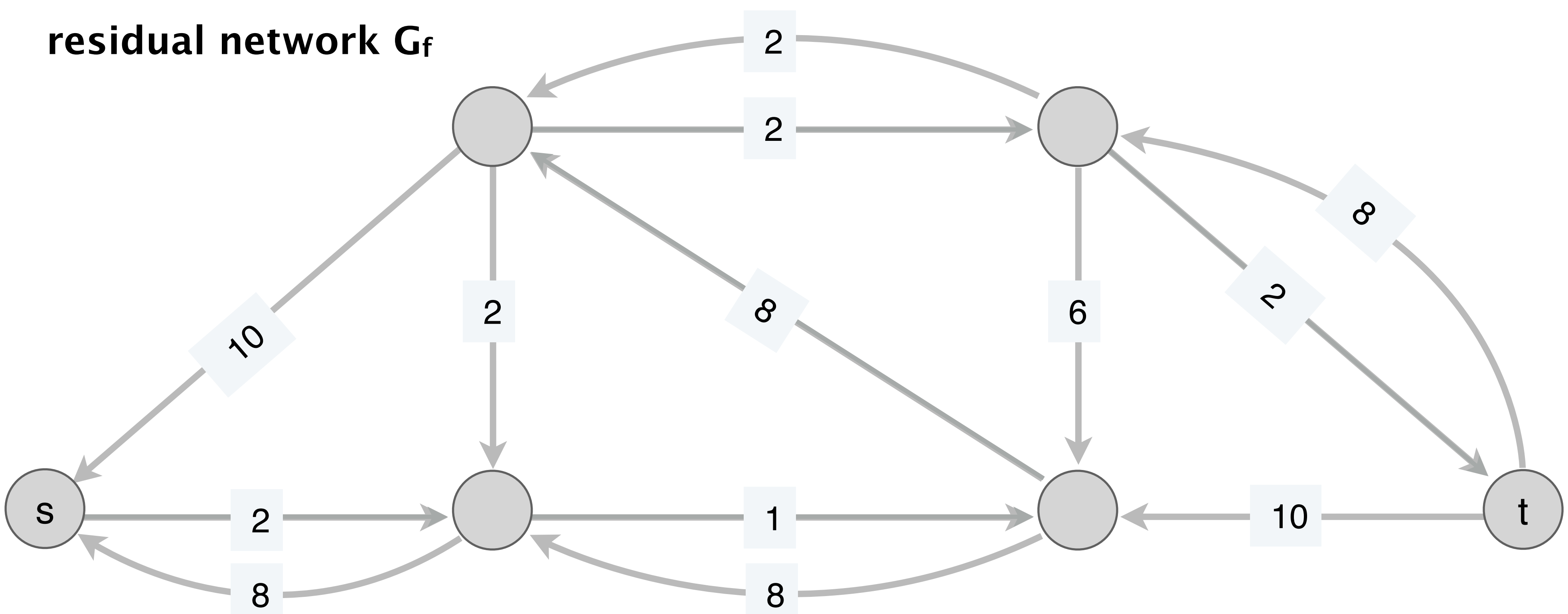
Ford-Fulkerson Example



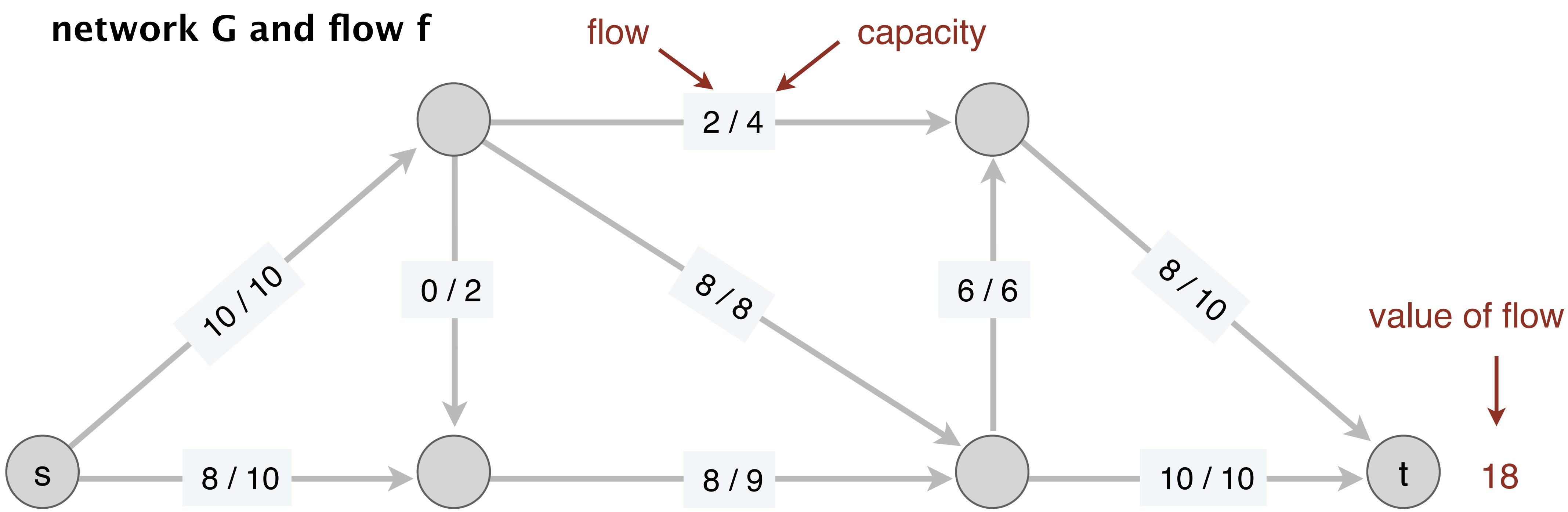


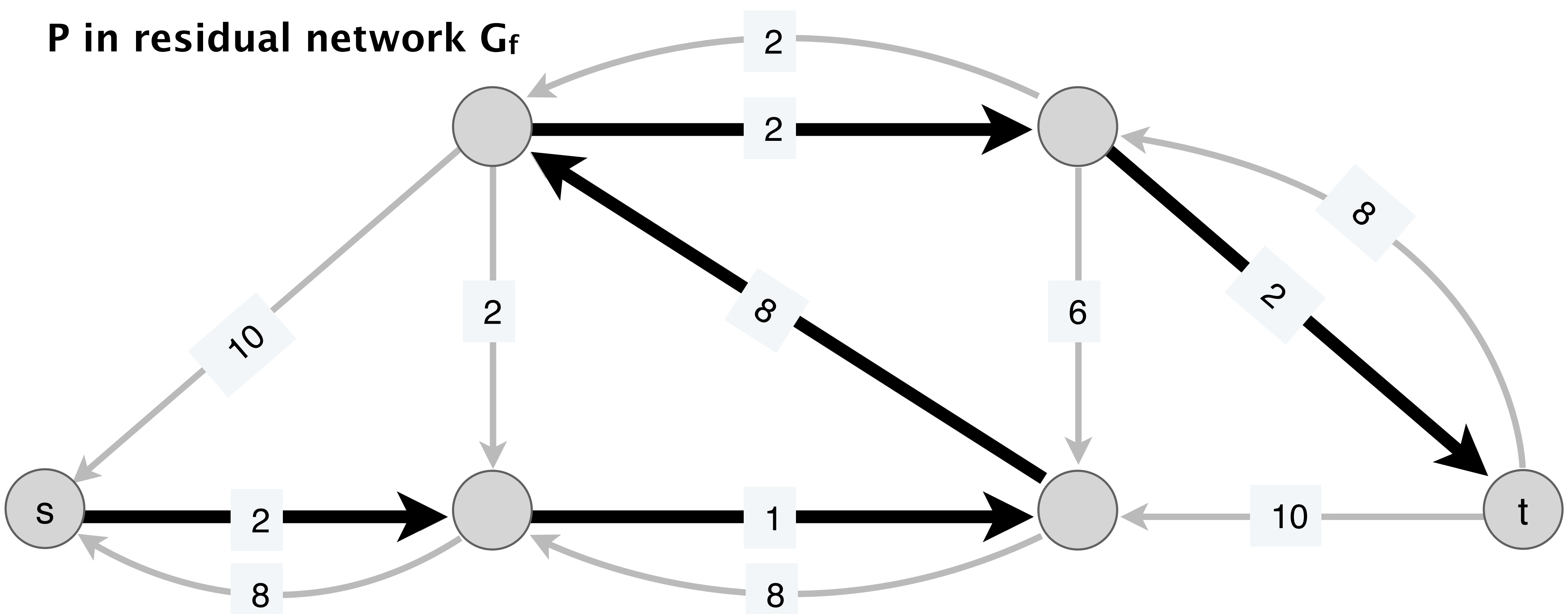
Ford-Fulkerson Example



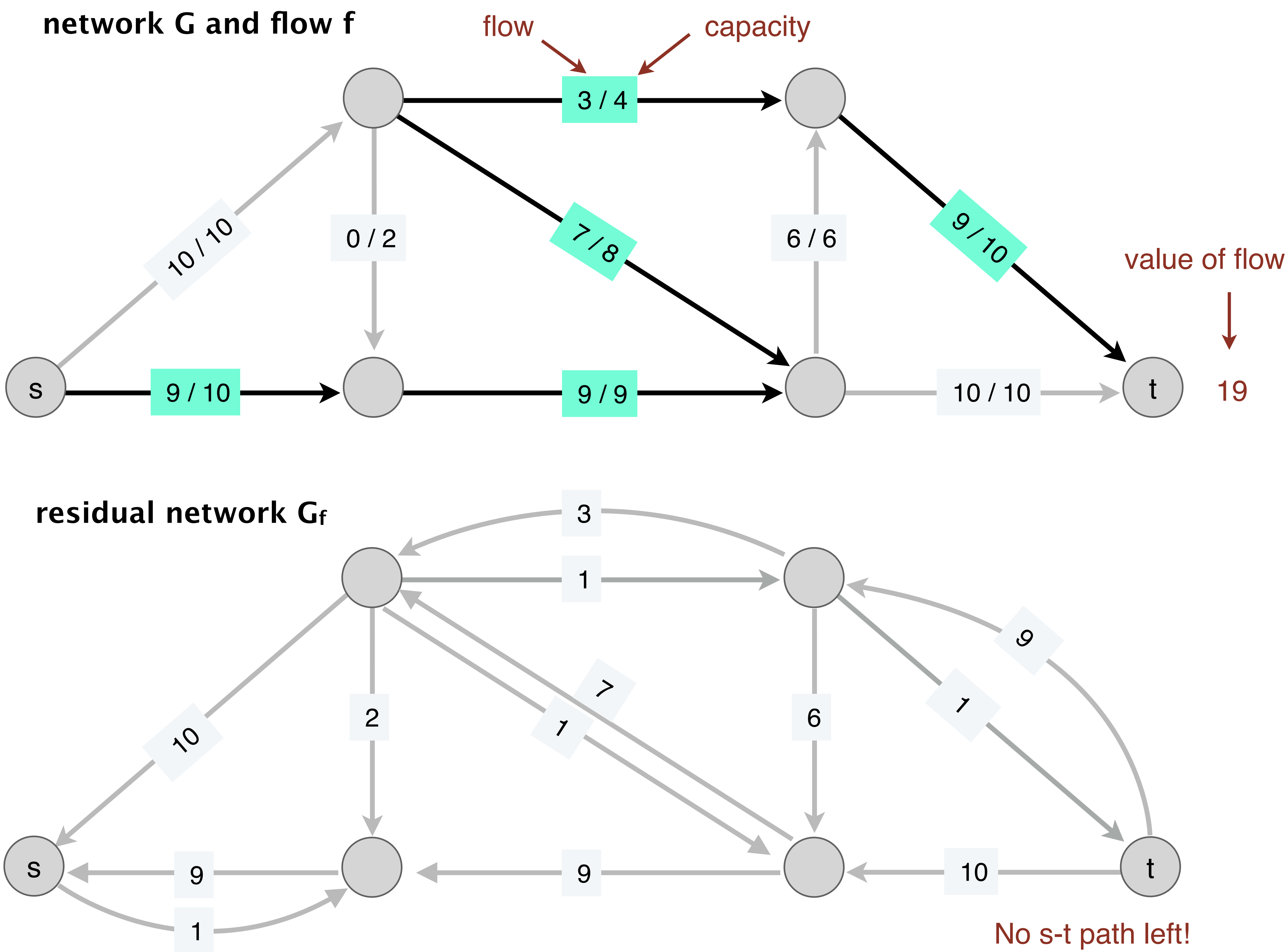


Ford-Fulkerson Example

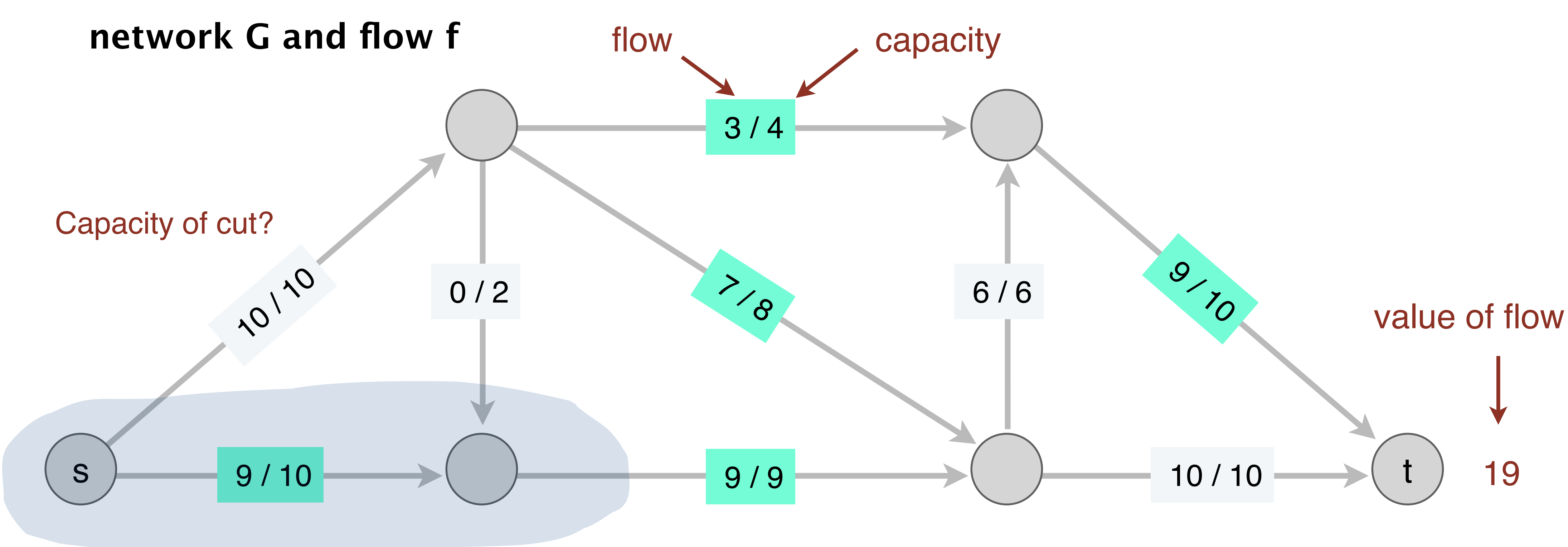


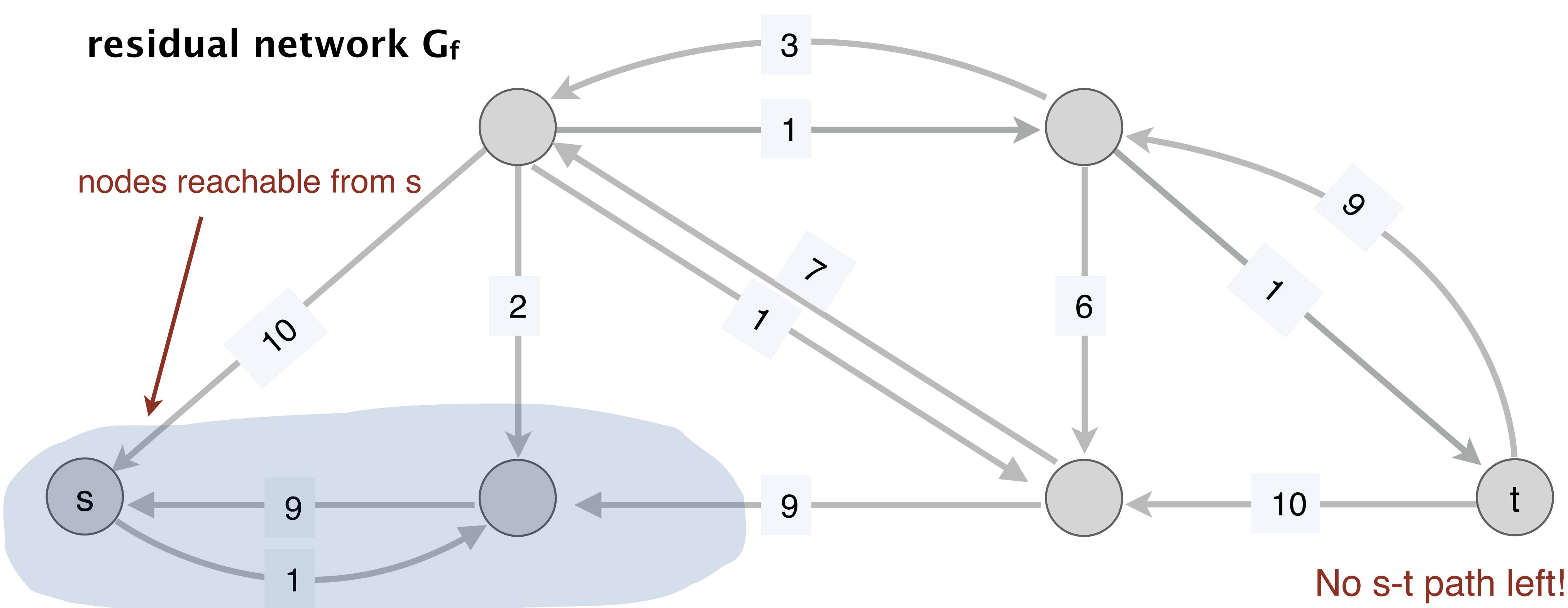


Ford-Fulkerson Example



Ford-Fulkerson Example





Analysis: Ford-Fulkerson

Analysis Outline (Things to Prove)

- Feasibility and value of flow:
  - Show that each time we update the flow, we are routing a feasible s-t flow through the network
  - And that value of this flow increases each time by that amount
- Optimality:
  - Final value of flow is the maximum possible
- Running time:
  - How long does it take for the algorithm to terminate?
- Space:
  - How much total space are we using?

Feasibility of 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 $f' \leftarrow \text{AUGMENT}(f, P)$, then f' is **a feasible flow**.
- **Proof**. Note, we 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: $f'(e) = f(e) + b$

    $$\leq f(e) \ + \ (c(e) - f(e)) \ = \ c(e)$$

  - If e is a backward edge: $f'(e) = f(e) - b$

    $$\geq f(e) \ - \ f(e) \ = \ 0$$

- Conservation constraint hold on any node in $u \in P$:
  - $f_{in}(u) = f_{out}(u)$, therefore $f'_{in}(u) = f'_{out}(u)$ for both cases

Value of Flow: Making Progress

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 $v(f') = v(f) + b$.

- **Proof**.
  - First edge $e \in P$ must be out of s in G_f
  - Observe that P is simple, so it 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 ∎
- Note. Means the algorithm makes forward progress each time!

We'll use this later to analyze the running time

Optimality

Ford-Fulkerson Optimality

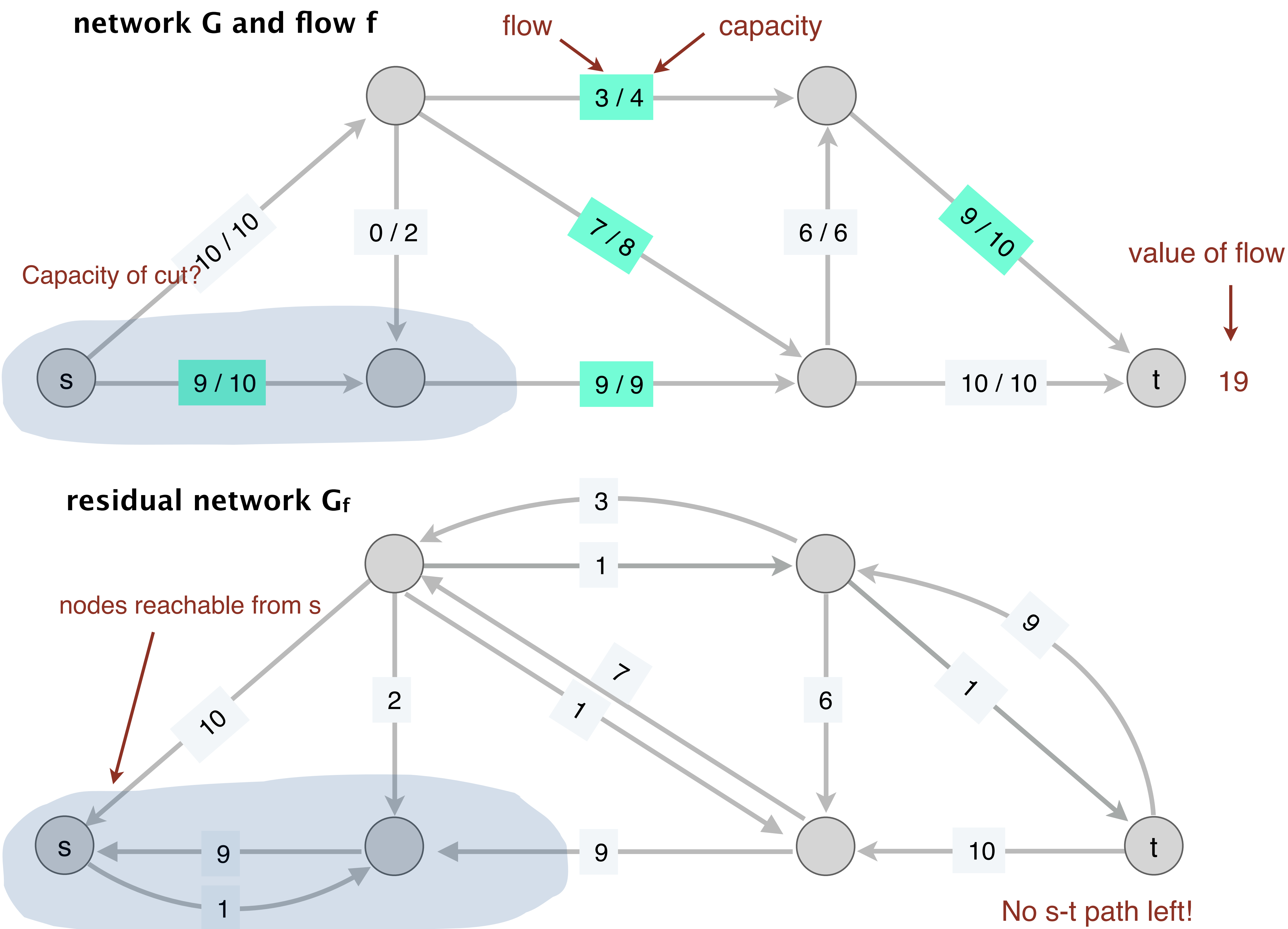
- **Recall**: If f is any feasible s-t flow and (S, T) is any s-t cut then $v(f) \leq c(S, T)$.
- We will show that the Ford-Fulkerson algorithm terminates in a flow that achieves optimality, that is,
  - Ford-Fulkerson finds a flow f^*, and there exists a cut (S^*, T^*) such that, $v(f^*) = c(S^*, T^*)$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

Ford-Fulkerson Optimality

Lemma. Let f be an 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?
  - Yes! $s \in S, t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
- Consider an edge $e = u \rightarrow v$ with $u \in S^*, v \in T^*$, then what can we say about $f(e)$?

Recall: Ford-Fulkerson Example



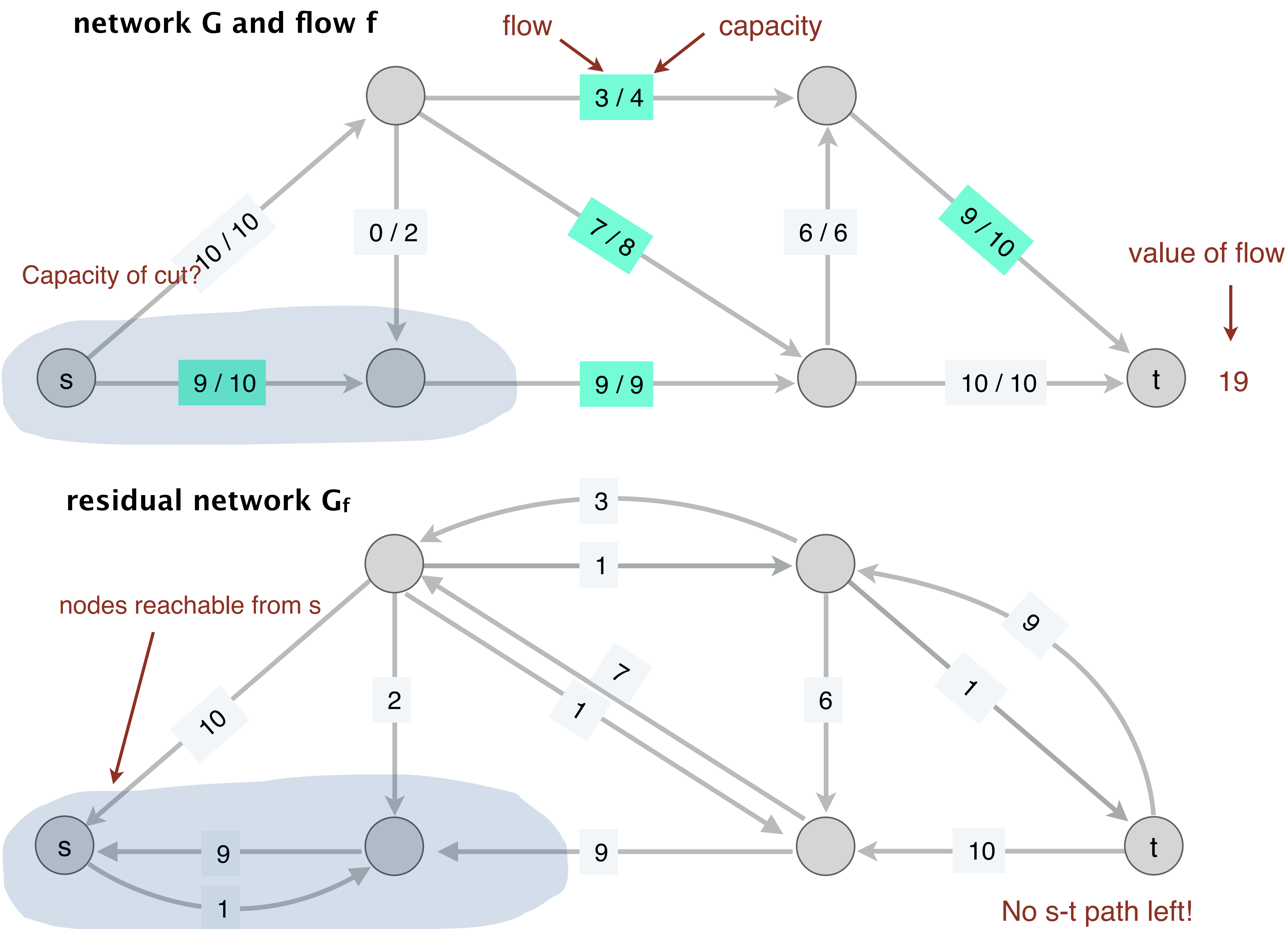
Ford-Fulkerson Optimality

- **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



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 $e = w \rightarrow v$ with $v \in S^*, w \in T^*$, then what can we say about $f(e)$?
  - $f(e) = 0$

Otherwise, there would have been a backwards edge in the residual graph

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 $G_f\}$, $T^* = V - S^*$
- 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.

Performance Questions:

- Does the while loop terminate?
- If it terminates, can we bound the number of iterations?
- What is the Big-O running time of the whole algorithm?

Ford-Fulkerson Running Time

Recall we proved that with each call to AUGMENT, we increase **value of the s-t flow** by $b = \text{bottleneck}(G_f, P)$

- **Assumption**. We assumed 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 $v(f) \leq nC$
- 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 Performance

```
FORD–FULKERSON(G)
_________________________________________________
FOREACH edge e ∈ E : f(e) ← 0.
G_f ← residual network of G with respect to flow f.
WHILE (there exists an s⤳t path P in G_f)
    f ← AUGMENT(f, P).
    Update G_f.
RETURN f.
```

We know there are $O(nC)$ iterations. How many operations per iteration?

- Cost to find an augmenting path in G_f?
- Cost to augment flow on path?
- Cost to update G_f?

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**. Time taken by each iteration:
- Finding 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
- Overall, $O(m)$ time per iteration ∎

[Digging Deeper] Polynomial time?

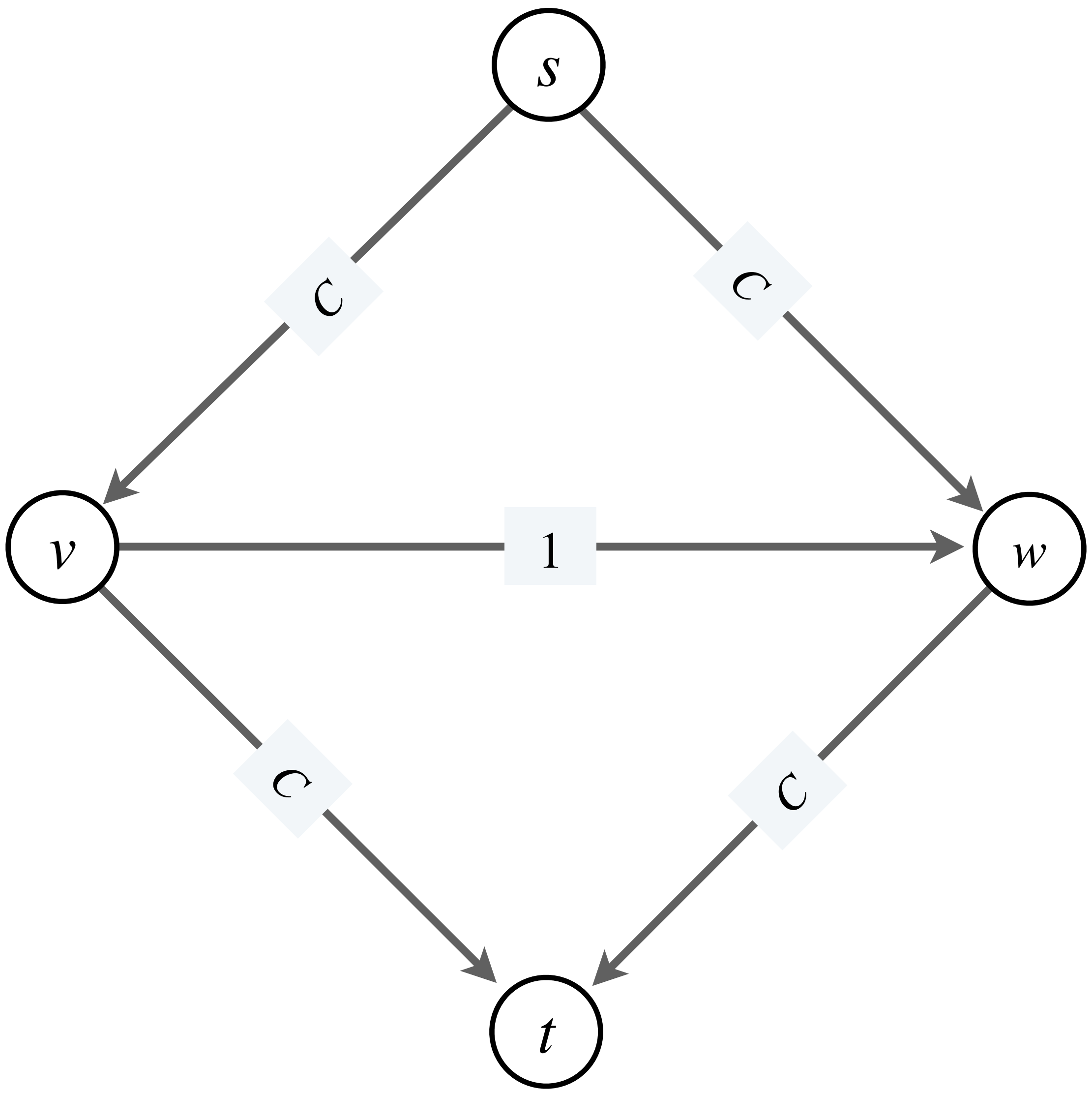
Question: Does the Ford-Fulkerson algorithm run in time polynomial *in the input size*?

- Running time is $O(nmC)$, where $C = \max_u c(s \to u)$
- What is the input size?
  - n vertices, m edges, m capacities
  - C represents the ***magnitude*** of the maximum capacity leaving the source node
  - How many bits to represent C?
    - $\log_2 C$
- Let us look at 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: $\Omega(n \log n + m \log n + m \log C)$ bits
- Running time: $O(nmC) = O(nm2^{\log_2 C})$
    - Exponential in the *size* of representing C
- Recall that such algorithms are called **pseudo-polynomial**
    - If the running time is polynomial in the **magnitude** but **not size** of an input parameter.
    - We saw this for knapsack as well!

Non-Integral Capacities?

Recall: our runtime analyst relied on integral capacities. What happens if they aren't?

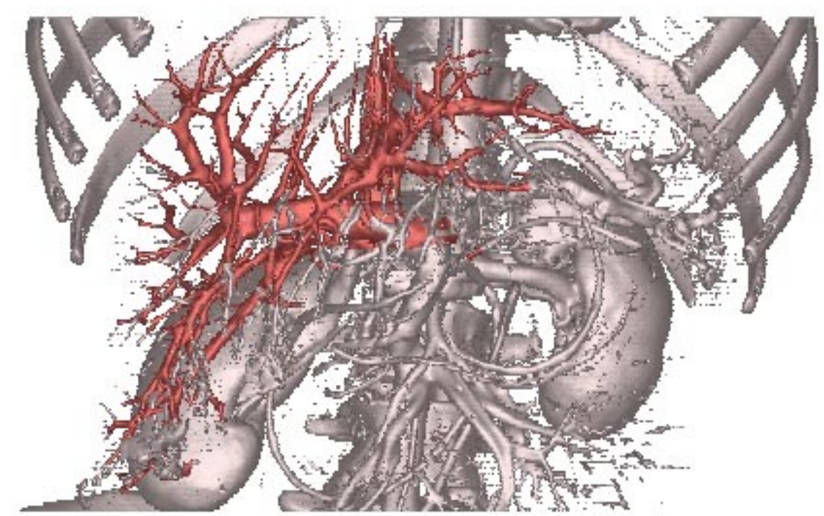
- If the capacities are rational, can just multiply to obtain a large integer
  - Increases running time, but Ford-Fulkerson analysis unchanged
- If capacities are irrational, Ford-Fulkerson can run infinitely!
  - Improvement at each step can be arbitrarily small
  - We can create bad instances where it doesn't terminate in finite steps

Applications of Network Flow:

Solving Problems by Reduction to Network Flows

Max-Flow Min-Cut Applications

- Data mining
- Bipartite matching
- Network reliability
- Image segmentation
- Baseball elimination
- Network connectivity
- Markov random fields
- Distributed computing
- Network intrusion detection
- **Many, many, more.**



liver and hepatic vascularization segmentation

Liver and hepatic vascularization segmentation using a Min-cut/Max-flow algorithm (S. Esneault, T. Pham, K. Torres)

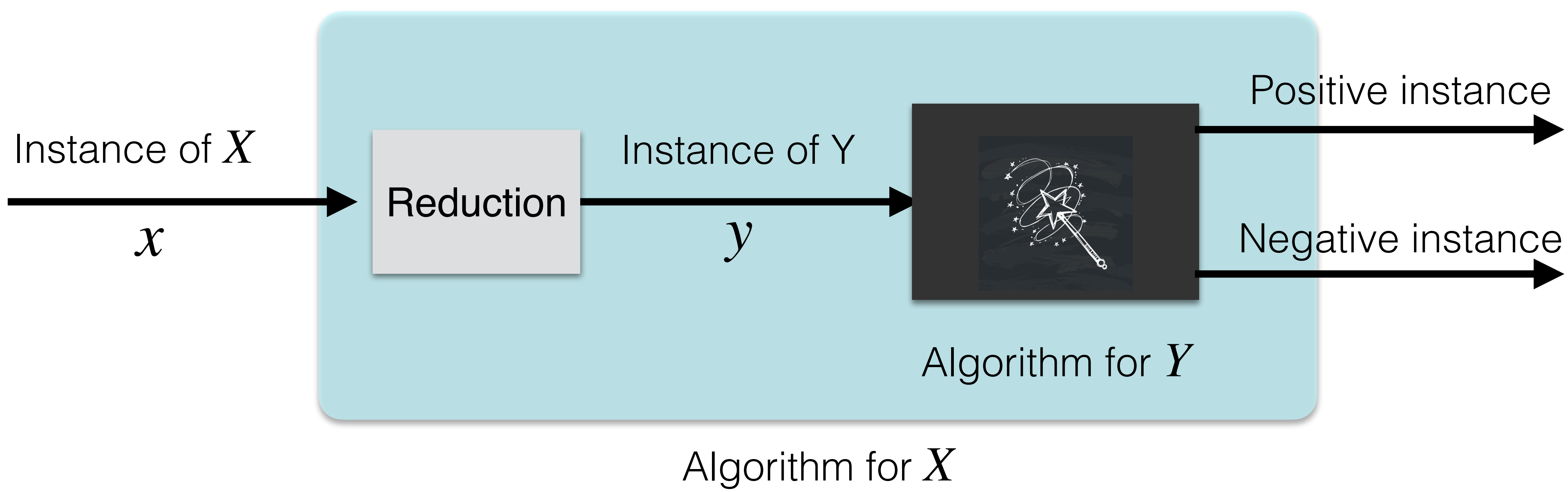
Reductions

- We will solve these problems by **reducing** them to a network flow problem
- We'll focus on the concept of **problem reductions**

Anatomy of Problem Reductions

At a high level, a problem X reduces to a problem Y if an algorithm for Y can be used to solve X

- **Reduction.** Convert an arbitrary instance x of X to a special instance y of Y such that there is a 1-1 correspondence between them



Anatomy of Problem Reductions

- **Claim.** x satisfies a property iff y satisfies a *corresponding* property
- Proving a reduction is correct: prove both directions
- x has a property (e.g. has matching of size k) $\implies$ y has a corresponding property (e.g. has a flow of value k)
- x does not have a property (e.g. does not have matching of size k) $\implies$ y does not have a corresponding property (e.g. does not have a flow of value k)
- Or equivalently (and this is often easier to prove):
  - y has a property (e.g. has flow of value k) $\implies$ x has a corresponding property (e.g. has a matching of value k)

Remaining Plan

We will explore one application of network flow in detail today

- Matching in bipartite graphs
- Matchings are super practical with many applications
- We have already seen one, can you remember?

Next meeting: another application reducible to network flow (baseball elimination)

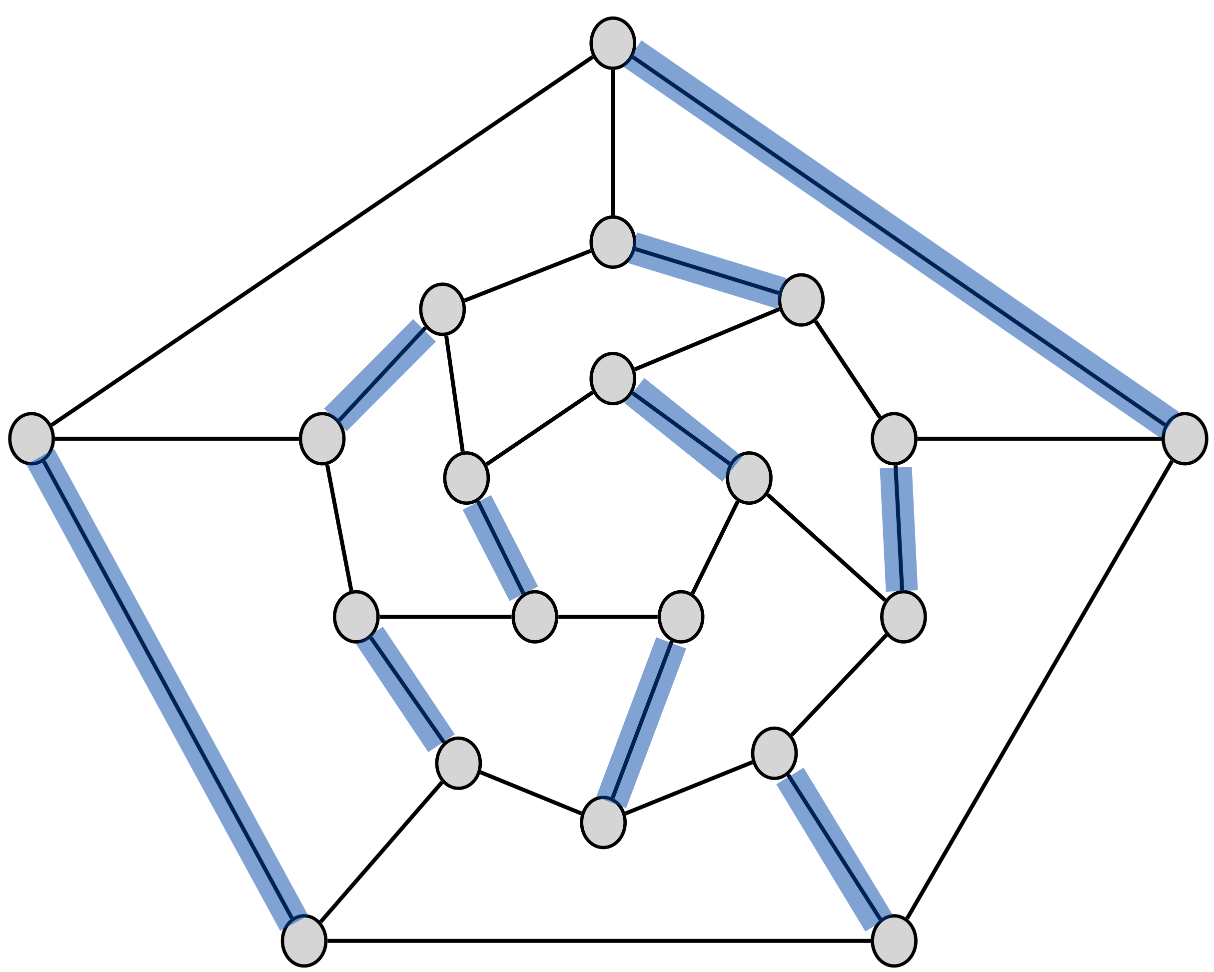
- More practice with reductions
- (Reductions will come in handy on our next topic too!)

Bipartite Matching

Review: Matching in Graphs

Definition. Given an undirected graph $G = (V, E)$, a matching $M \subseteq E$ of G is a subset of edges such that no two edges in M are incident on the same vertex.

- Said differently, a node appears in at most one edge in M



Review: Matching in Graphs

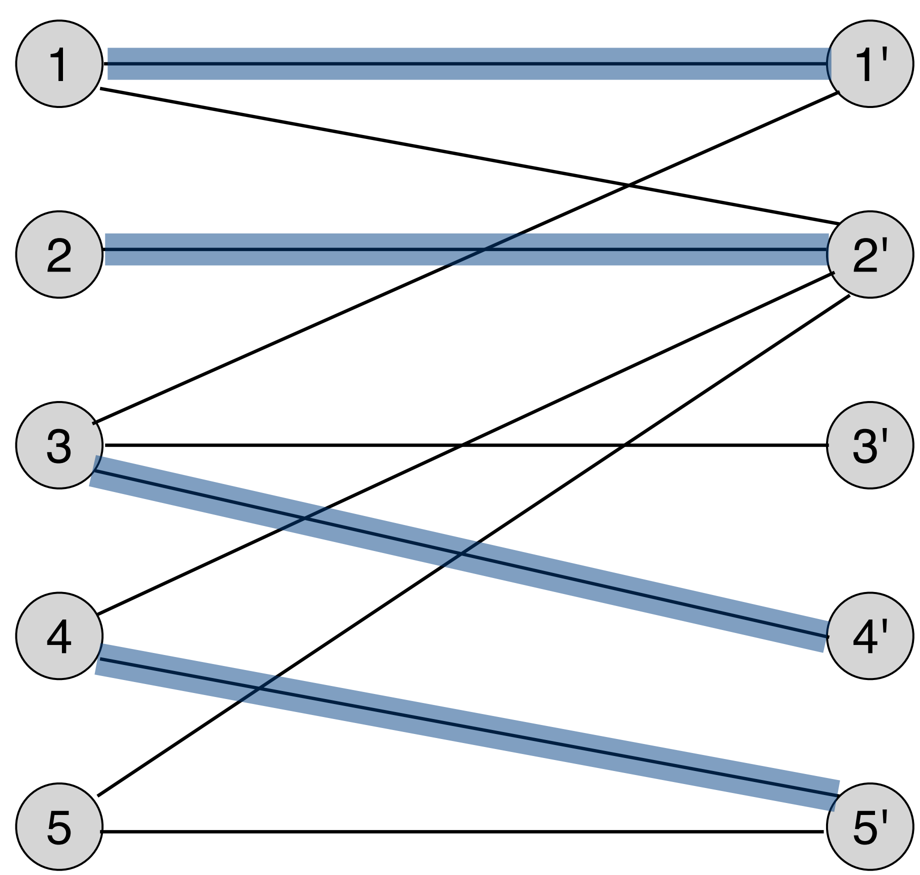
A **perfect matching** matches all nodes in G

- **Max matching problem.** Find a matching of maximum cardinality for a given graph
  - That is, a matching with maximum number of edges
  - **Observation**: If it exists, a perfect matching is maximum!

Review: Bipartite Graphs

A graph is **bipartite** if its vertices can be partitioned into two subsets X, Y such that every edge $e = (u, v)$ connects $u \in X$ and $v \in Y$

- **Bipartite matching problem.** Given a bipartite graph $G = (X \cup Y, E)$ find a maximum matching.



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)