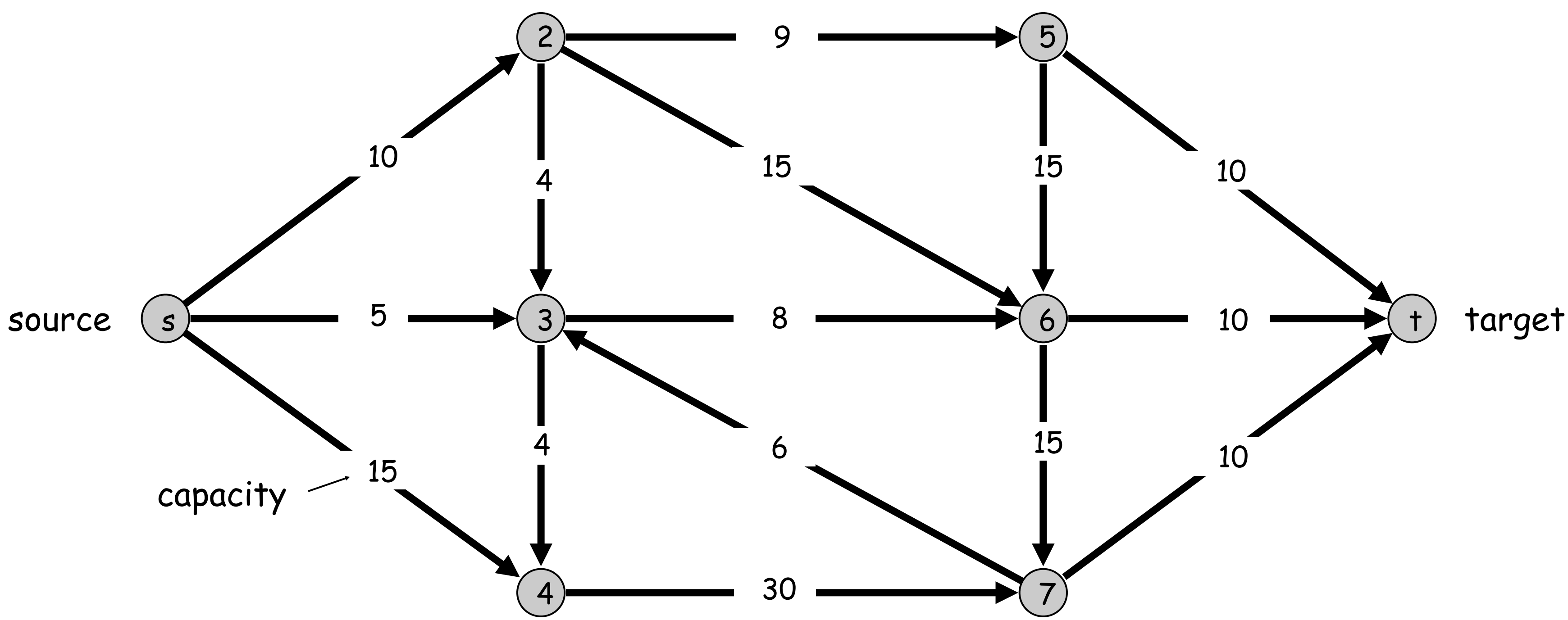
# Minimum Cut Problem

Flow network.

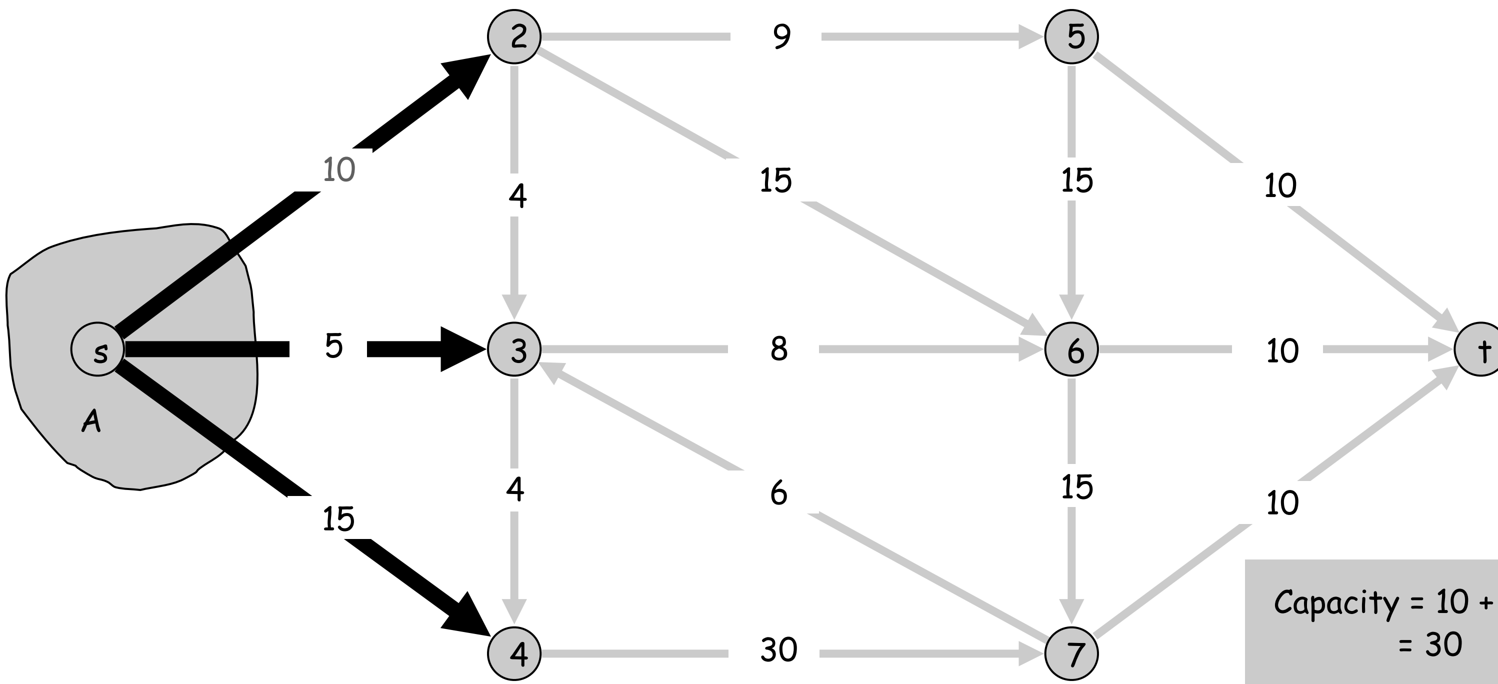
- Abstraction for substance flowing through the edges.
- G = (V, E) = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = target.
- c(e) = capacity of edge e.



# Cuts

**Def.** An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

**Def.** The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



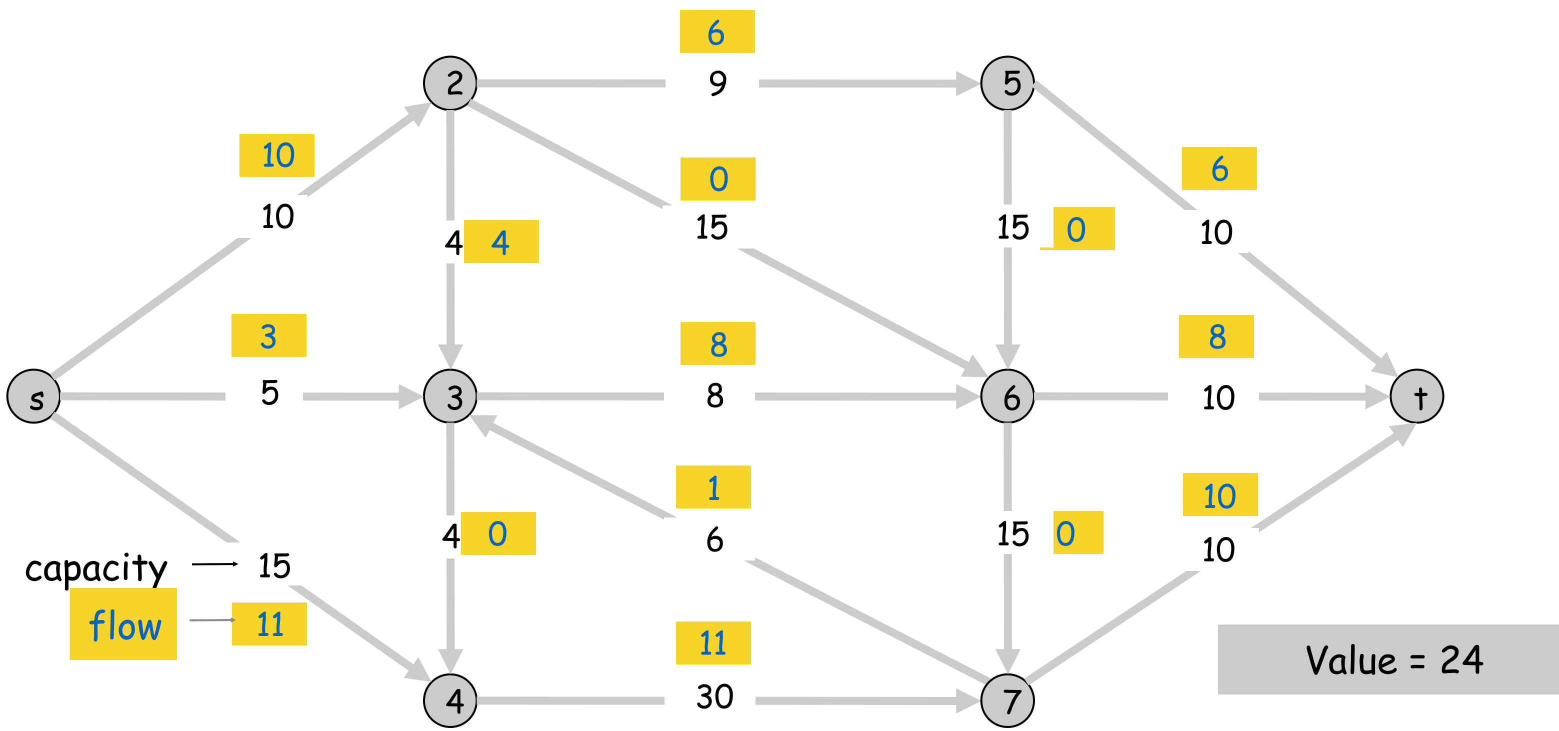

Capacity = 10 + 5 + 15
= 30

# Flows

**Def.** An **s-t flow** is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (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)

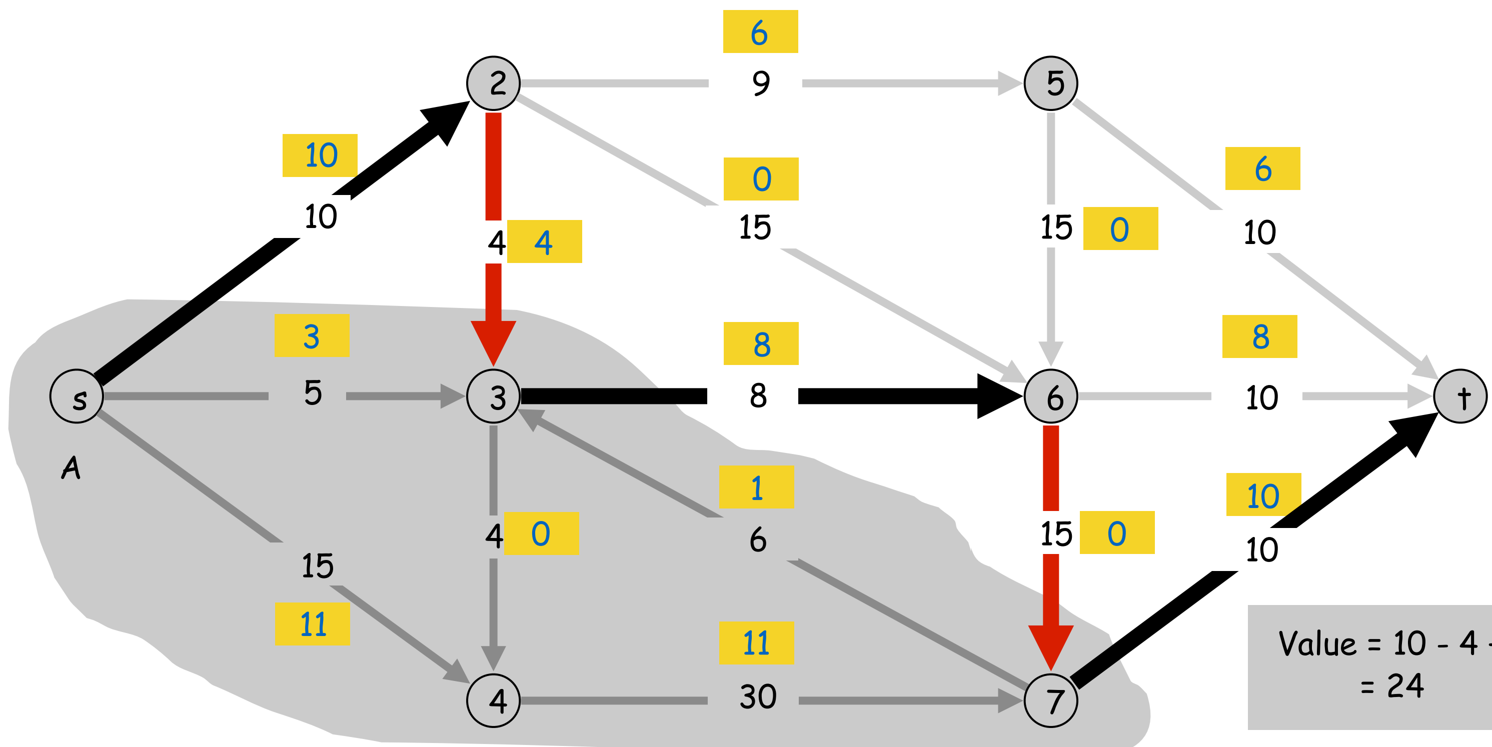
**Def.** The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$ .



# Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

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




Value = 10 - 4 + 8 - 0 + 10
= 24

# Flows and Cuts

**Flow value lemma.** Let f be any flow, and let (A, B) be any s-t cut. Then

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

**Pf.**

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

by flow conservation, all terms except v = s are 0 ➡

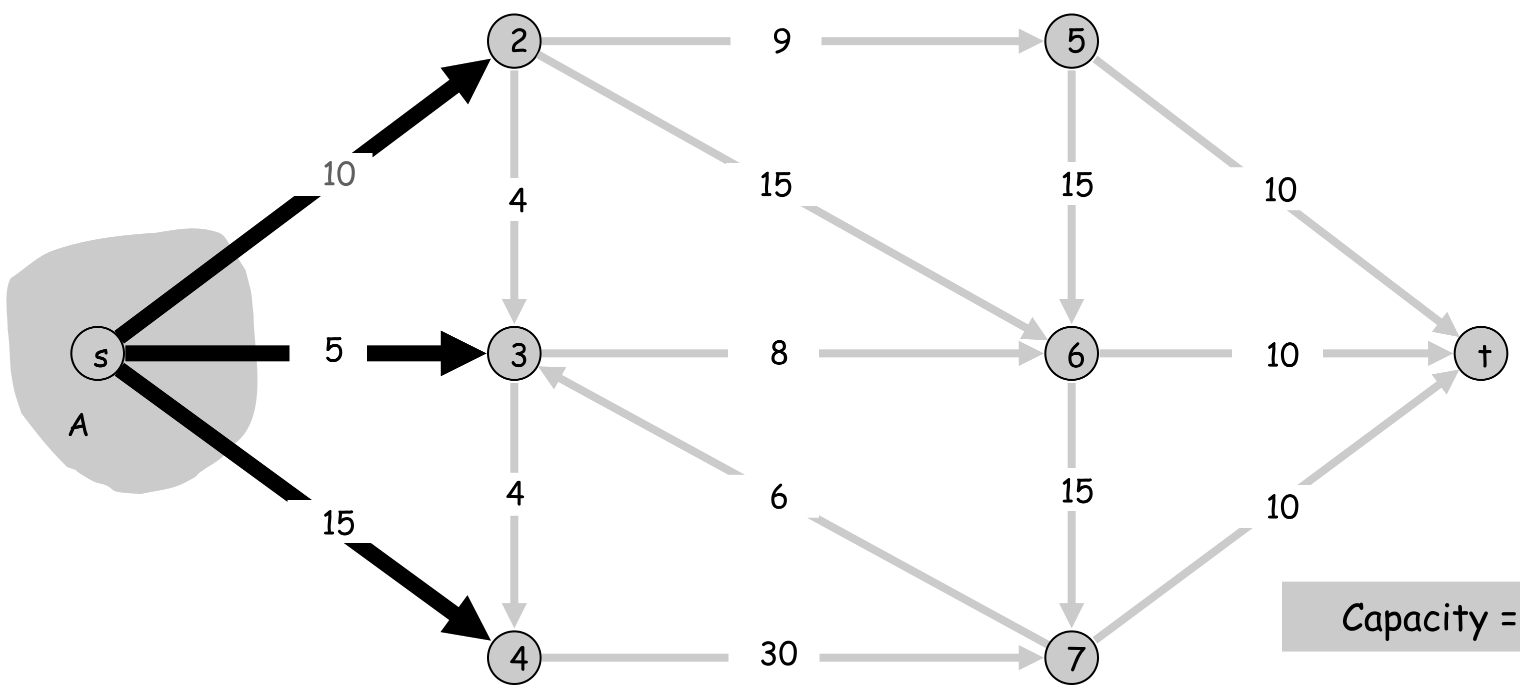
$$= \sum_{v \in A} \Big( \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \Big)$$

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

# Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



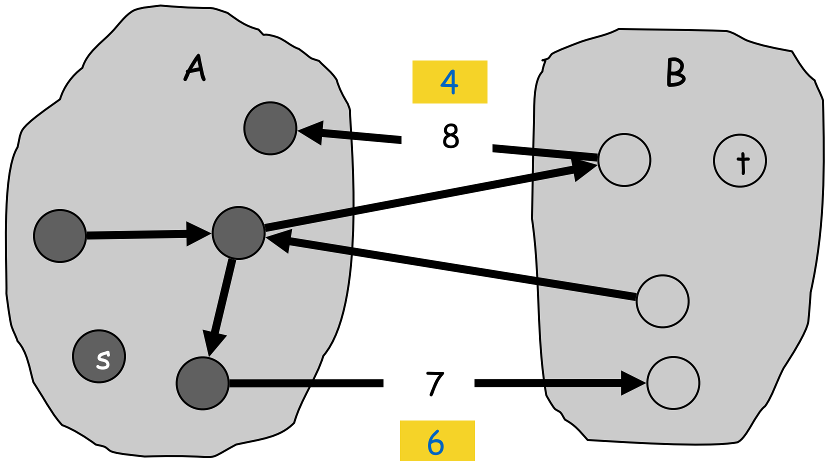

Capacity = 30

# Flows and Cuts

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have v(f) ≤ cap(A, B).

Pf.

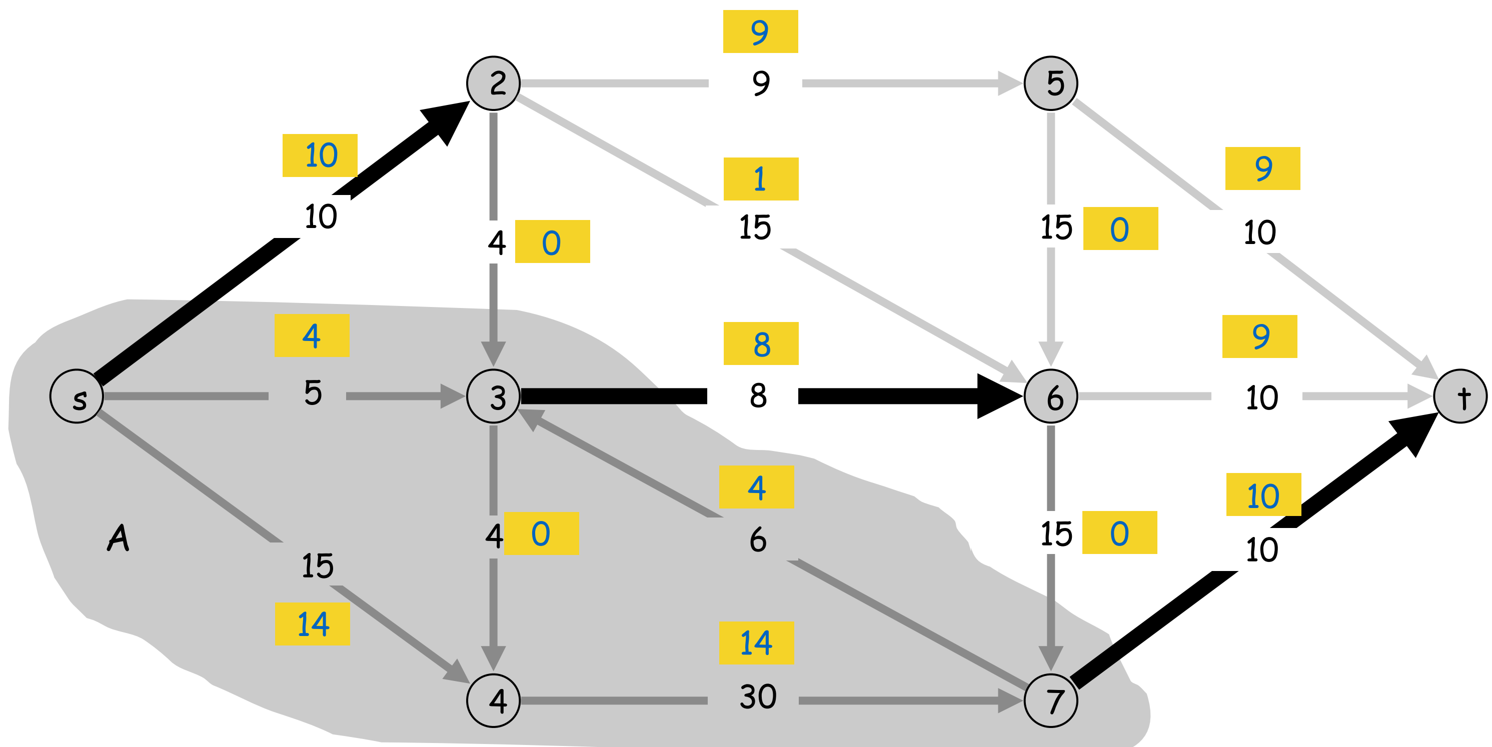
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&\le \sum_{e \text{ out of } A} f(e) \\
&\le \sum_{e \text{ out of } A} c(e) \\
&= \operatorname{cap}(A, B) \quad \blacksquare
\end{aligned}
$$



# Certificate of Optimality

Corollary. Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

Value of flow = 28
Cut capacity = 28 ⇒ Flow value ≤ 28

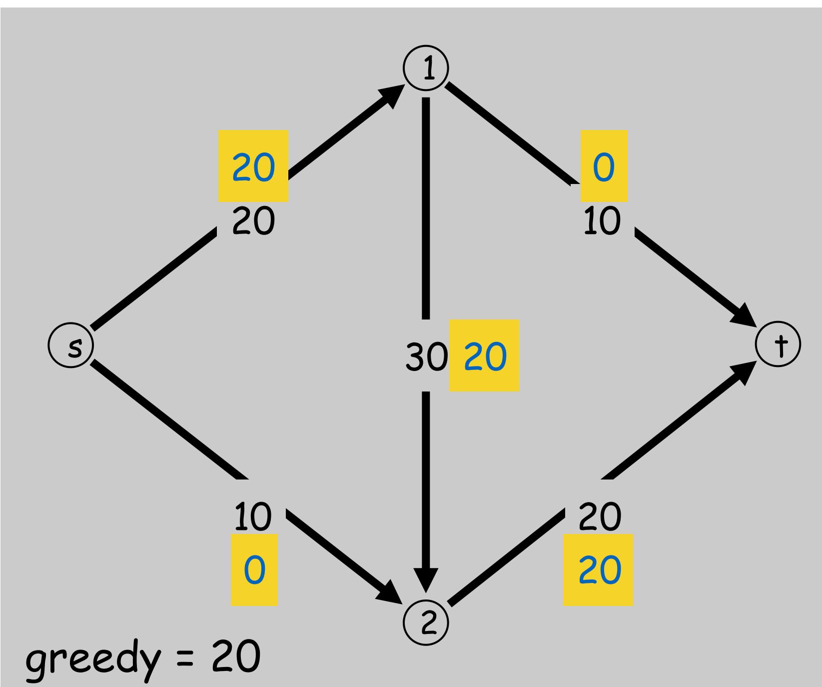


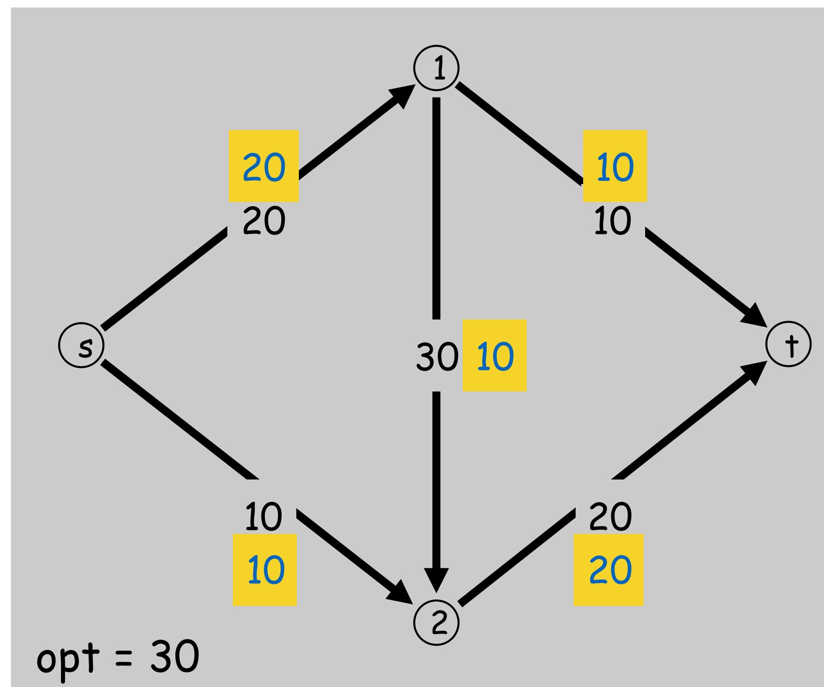
# Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path $P$ where each edge has $f(e) < c(e)$.
- Augment flow along path $P$.
- Repeat until you get stuck.

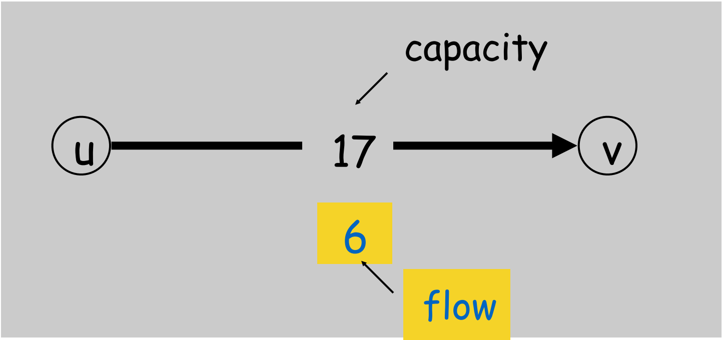
locally optimality $\not\Rightarrow$ global optimality



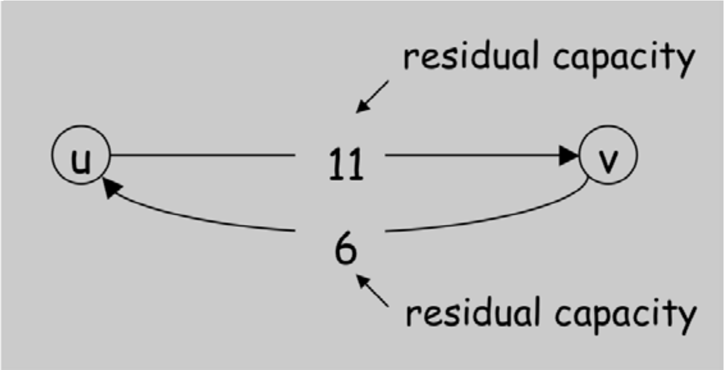
# Residual Graph

**Original edge:** $e = (u, v) \in E$.
- Flow $f(e)$, capacity $c(e)$.



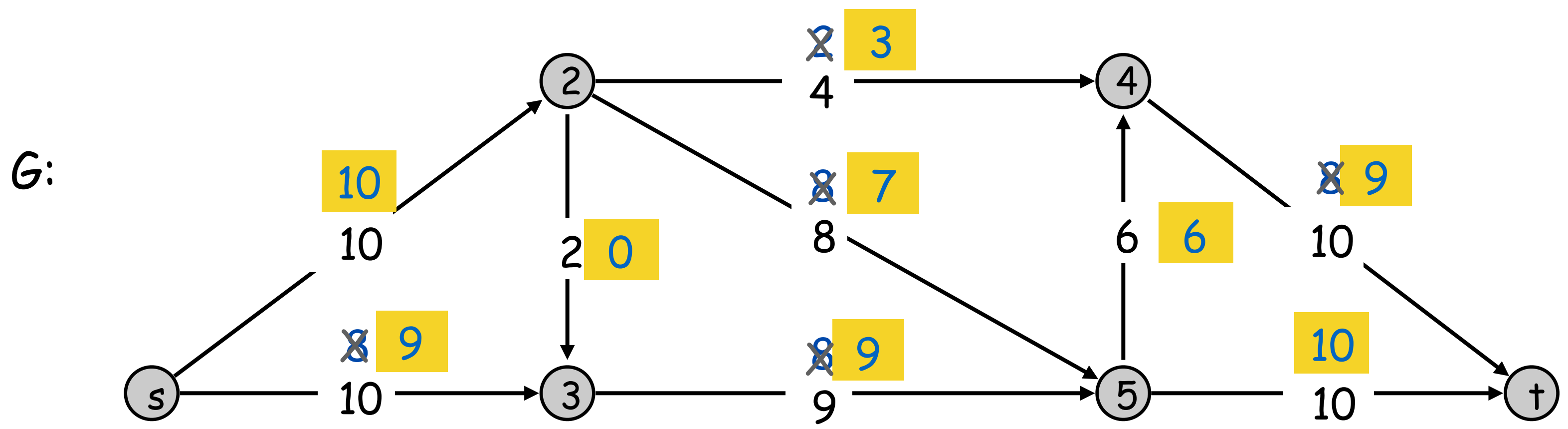

**Residual edge.**
- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

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

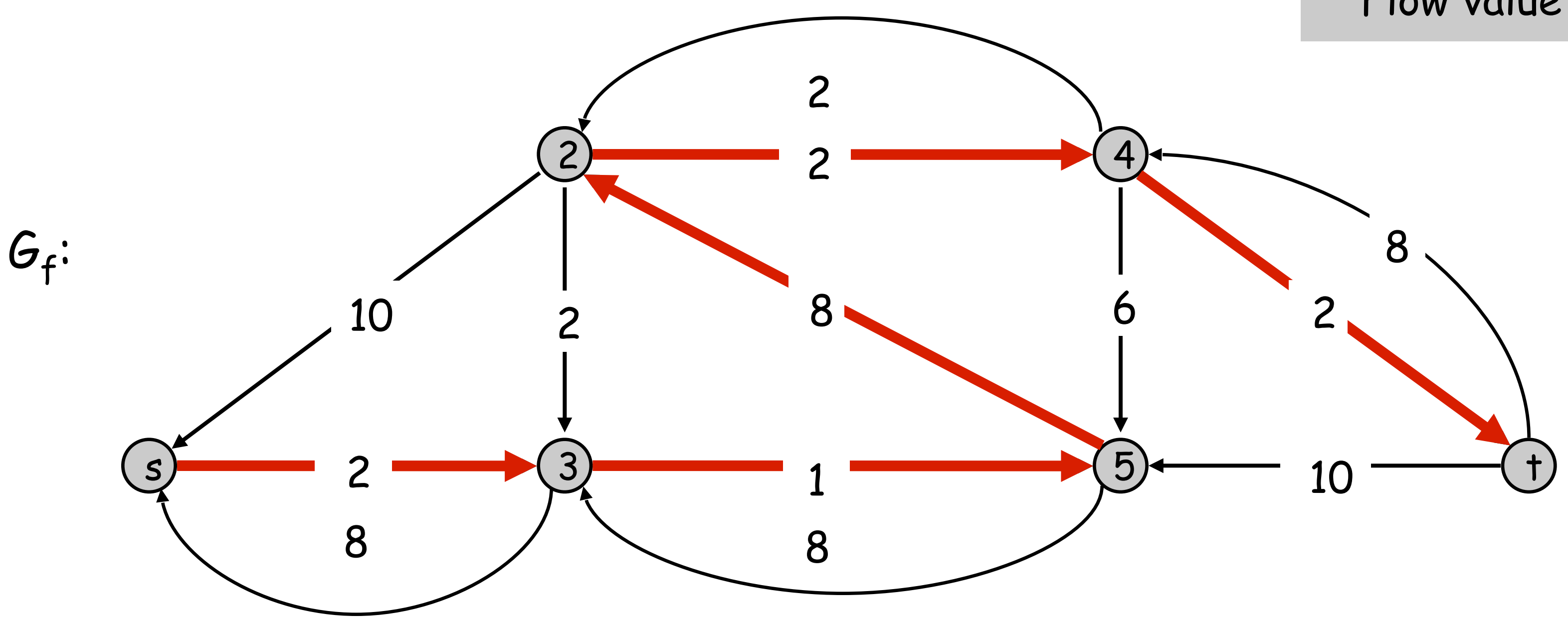



**Residual graph:** $G_f = (V, E_f)$.
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

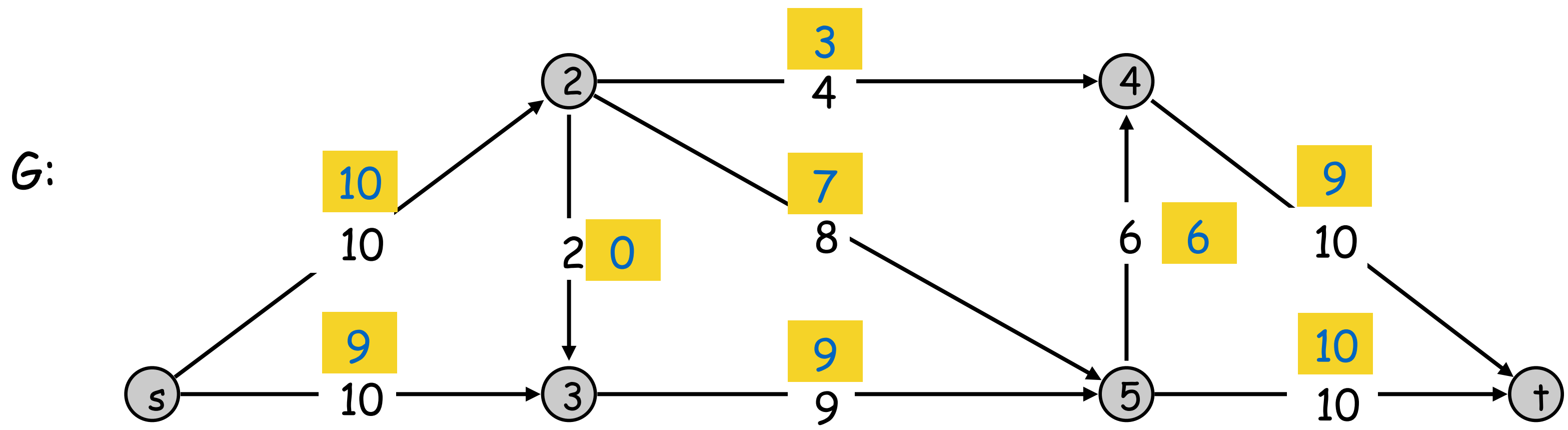
# Ford-Fulkerson Algorithm



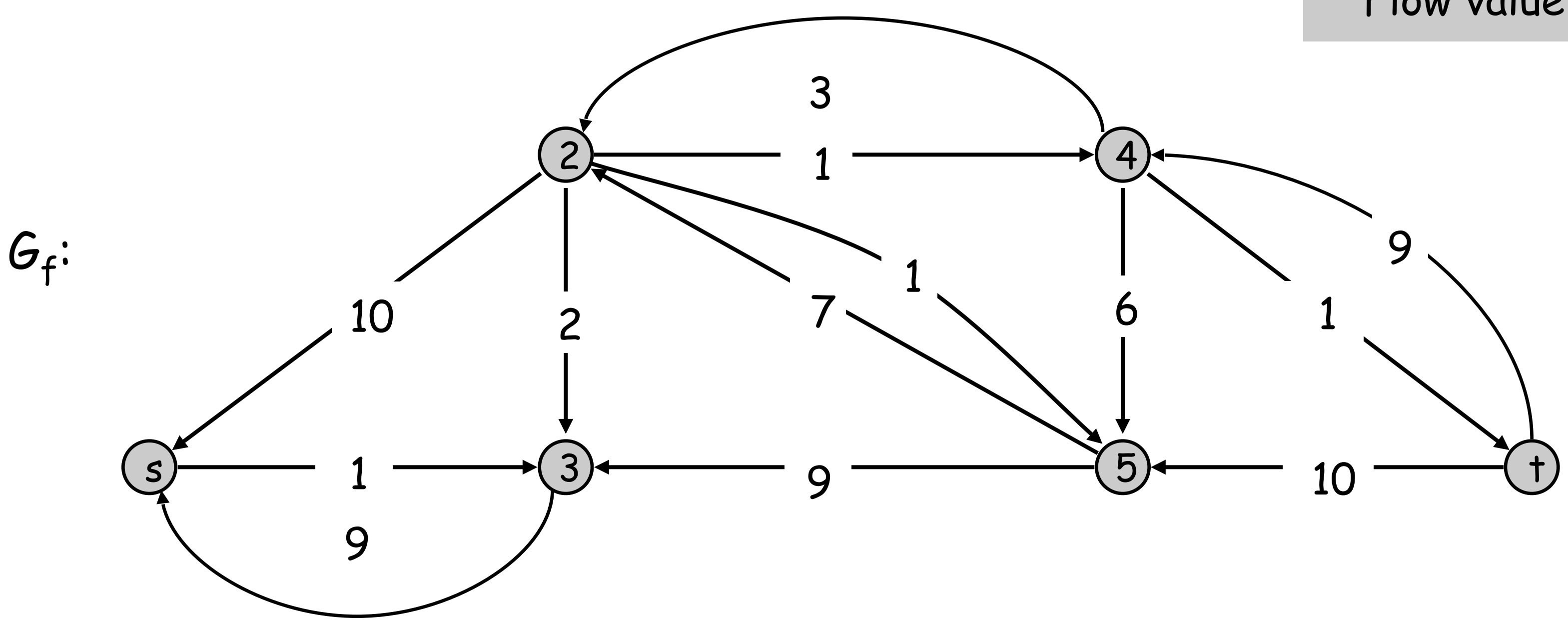

Flow value = 18



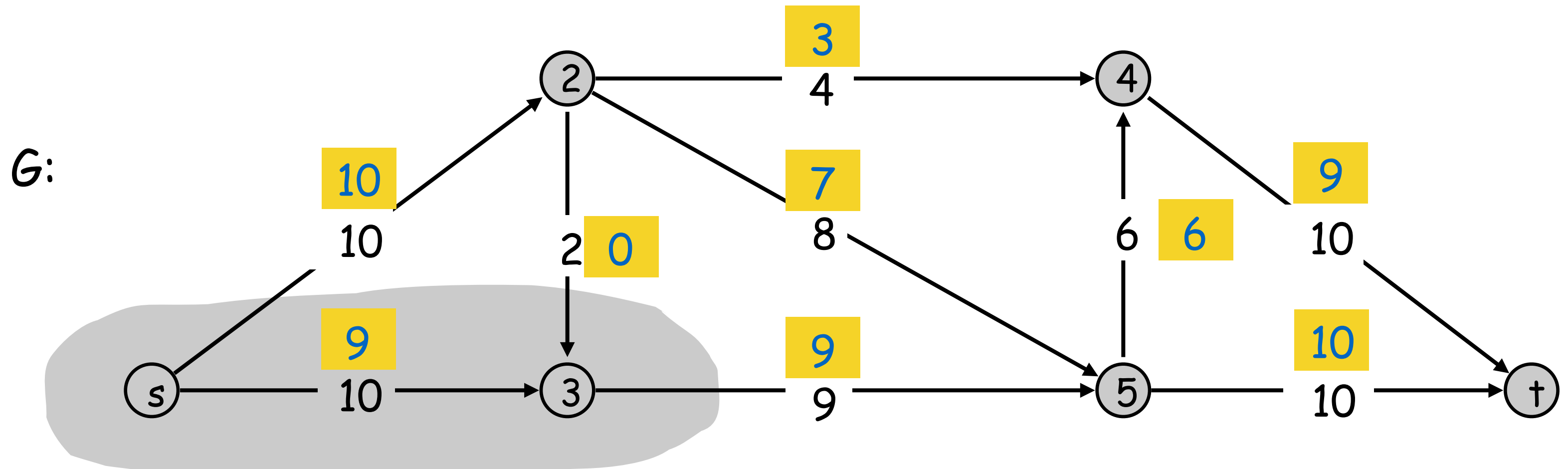
# Ford-Fulkerson Algorithm




Flow value = 19

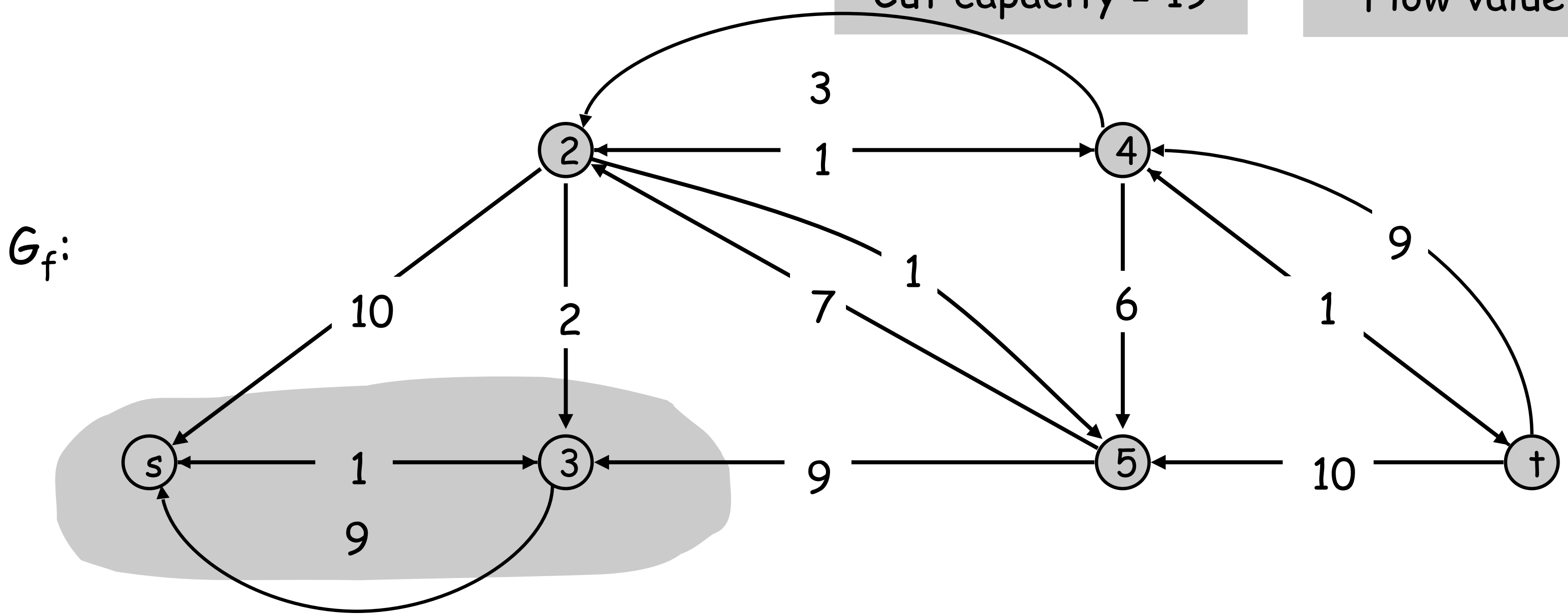


# Ford-Fulkerson Algorithm




Cut capacity = 19

Flow value = 19



# Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P,c,f)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b
      else       f(eR) ← f(eR) - b
   }
   return f
}
```

min residual capacity of edge on P

forward edge

reverse edge

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P in Gf) {
      f ← Augment(f, c, P)
      update Gf (along path P)
   }
   return f
}
```

# Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow iff 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.

Proof strategy. We prove both simultaneously by showing the following are equivalent:

(i) There exists a cut (A, B) such that v(f) = cap(A, B).
(ii) Flow f is a max flow.
(iii) There is no augmenting path relative to f.

(i) ⇒ (ii) This was the corollary to weak duality lemma.

(ii) ⇒ (iii) We show contrapositive: Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

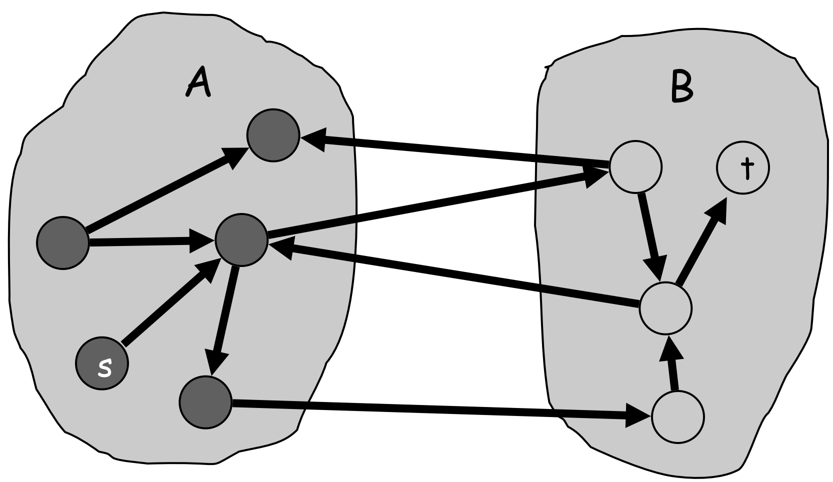
# Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A, s $\in$ A.
- By definition of $G_f$, t $\notin$ A.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \\ &= cap(A, B) \quad \blacksquare \end{aligned}$$

(otherwise connects outside A)




original network

# Running Time

Assumption. All capacities are integers between 1 and $C$.

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

Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations.
Pf. Each augmentation increases value by at least 1. ▪

Corollary. If $C = 1$, Ford-Fulkerson runs in $O(mn)$ time.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value $f(e)$ is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪

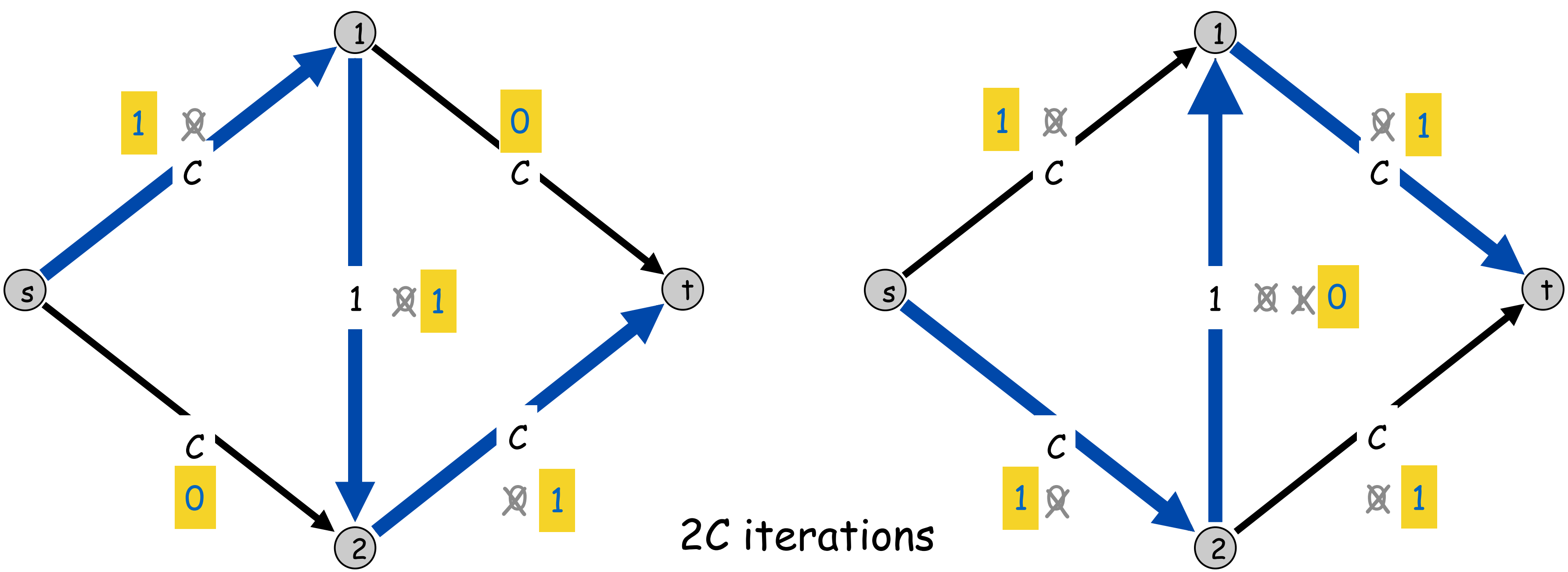
# 7.3 Choosing Good Augmenting Paths

# Ford-Fulkerson: Exponential Number of Augmentations

**Q.** Is generic Ford-Fulkerson algorithm polynomial in input size?

m, n, and log C

**A.** No. If C=largest capacity, then algorithm can take ≥C iterations.




2C iterations

# Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

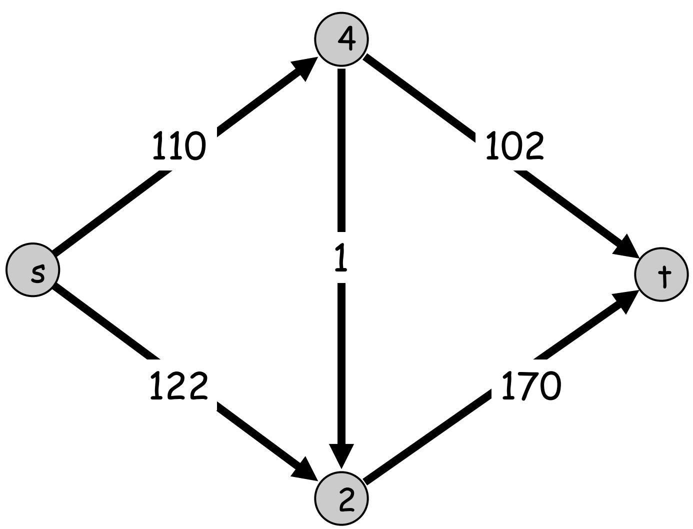
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

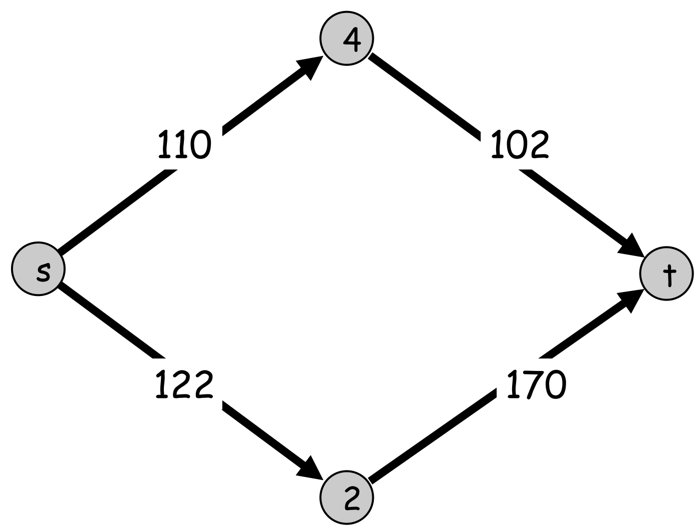
# Capacity Scaling

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 $\Delta$.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least $\Delta$.




$G_f$




$G_f(100)$

# Capacity Scaling

i think first Scale to find better aug
paths, then run Ford-Fulker

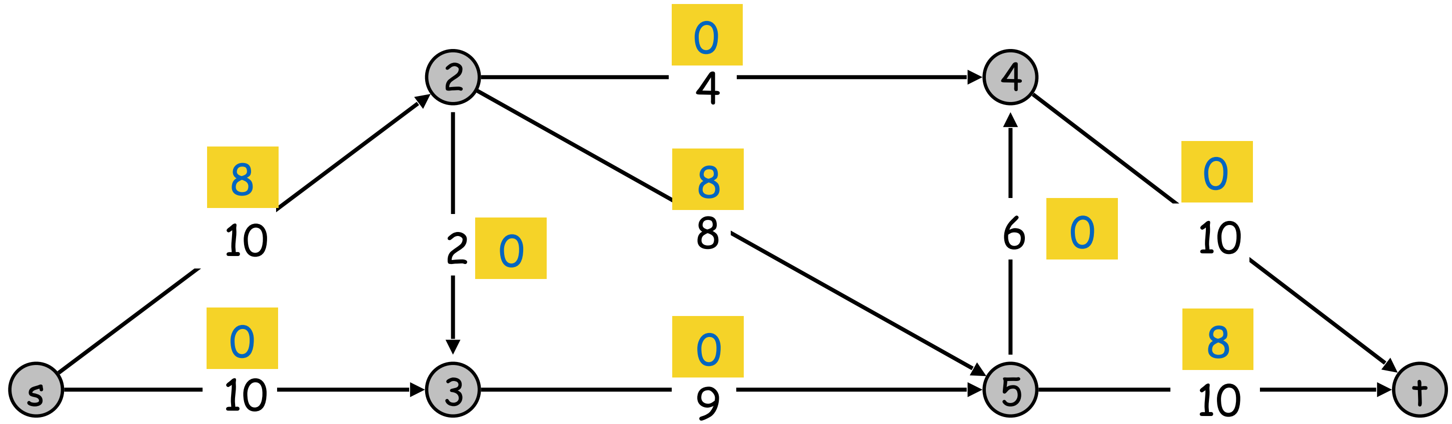
chose by max(min(edge in path))
of all paths

```
Scaling-Max-Flow(G, s, t, c, C) {
   foreach e ∈ E  f(e) ← 0
   Δ ← largest power of 2 ≤ C
   G_f ← residual graph


   while (Δ ≥ 1) {
      G_f(Δ) ← Δ-residual graph
      while (there exists augmenting path P in G_f(Δ)) {
         f ← augment(f, c, P)
         update G_f(Δ) (along P)
      }
      Δ ← Δ / 2
   }
   return f
}
```

# Ford-Fulkerson Algorithm with Capacity Scaling

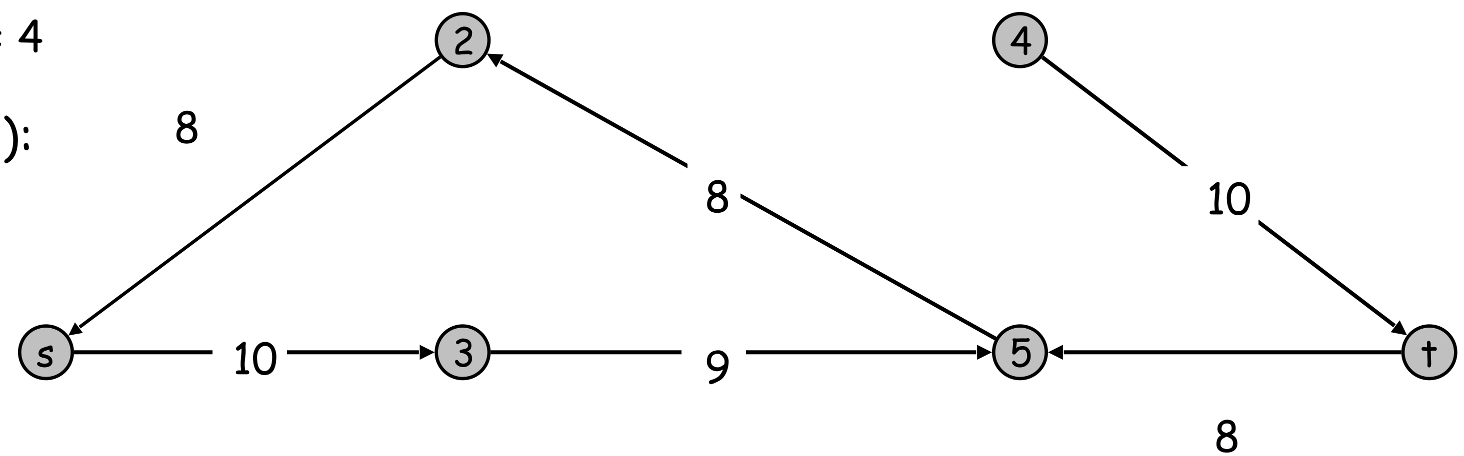
G:




Flow value = 8

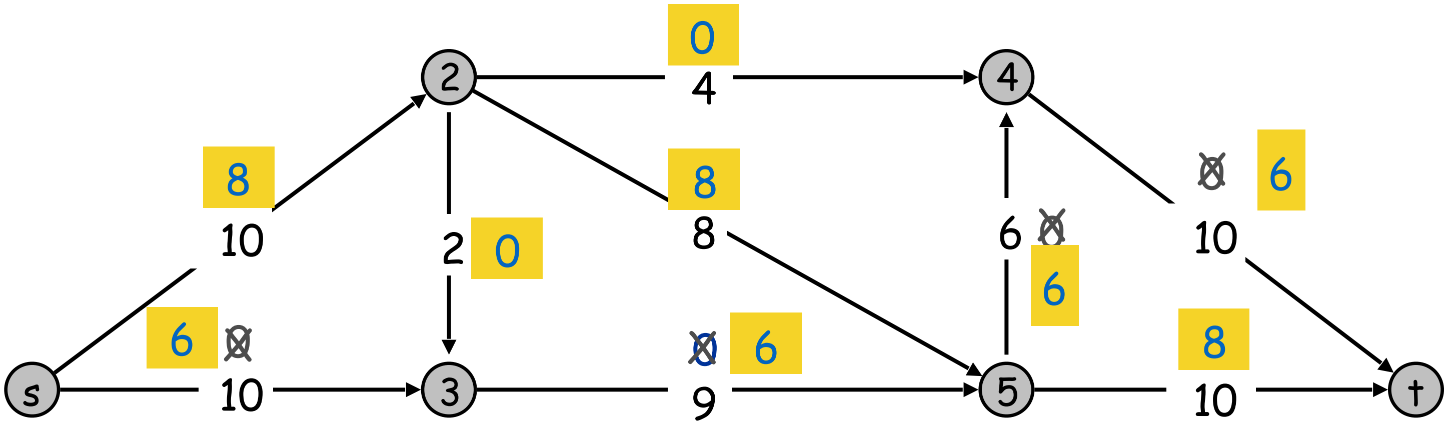
Δ = 4

$G_f(8)$:



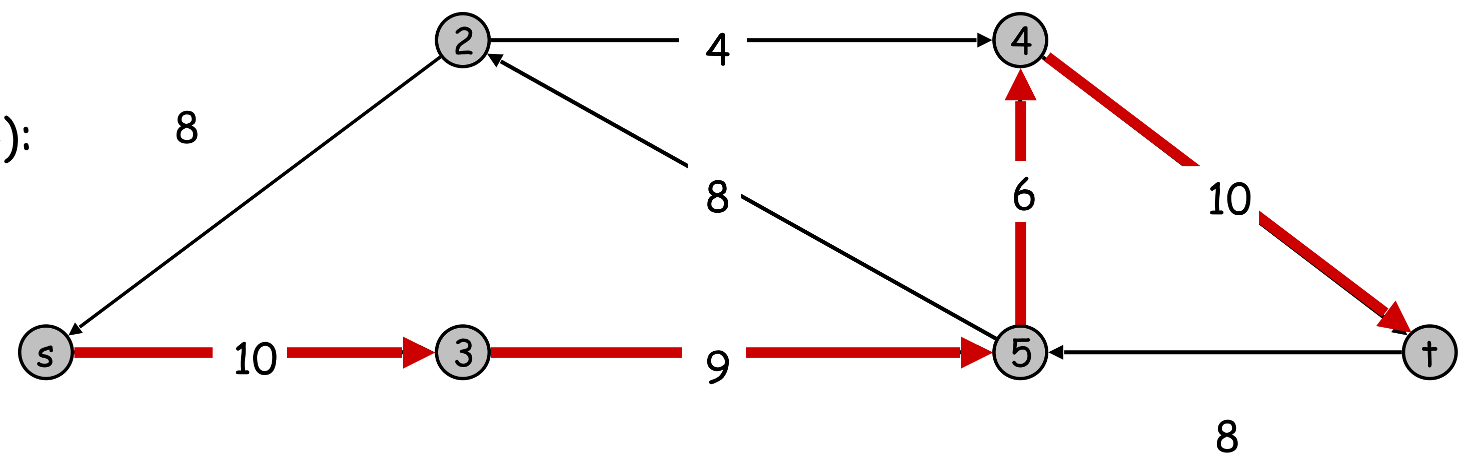
# Ford-Fulkerson Algorithm with Capacity Scaling

G:



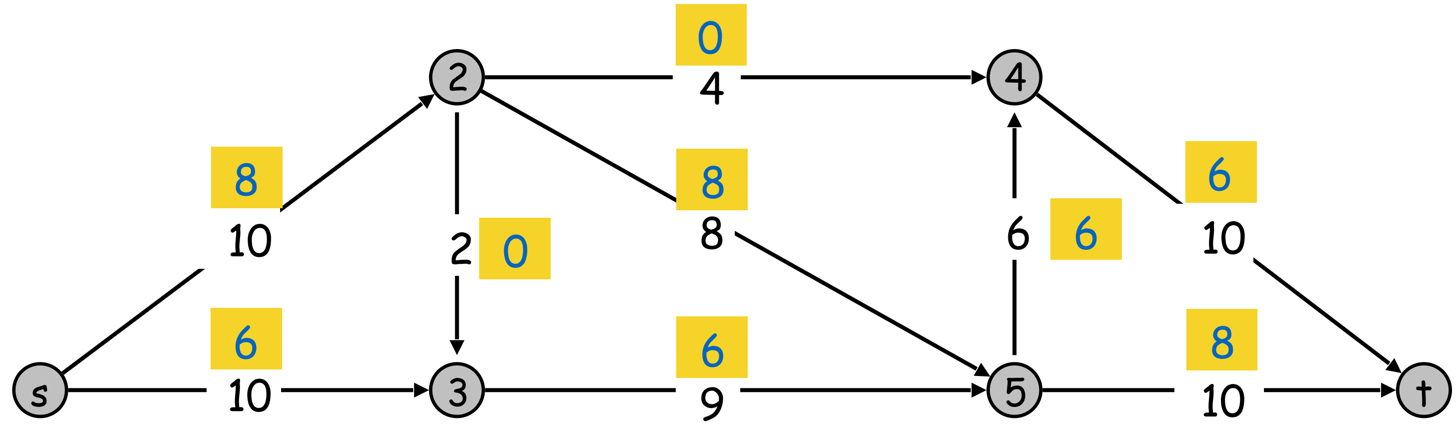

Flow value = 8

$G_f(4)$:



# Ford-Fulkerson Algorithm with Capacity Scaling

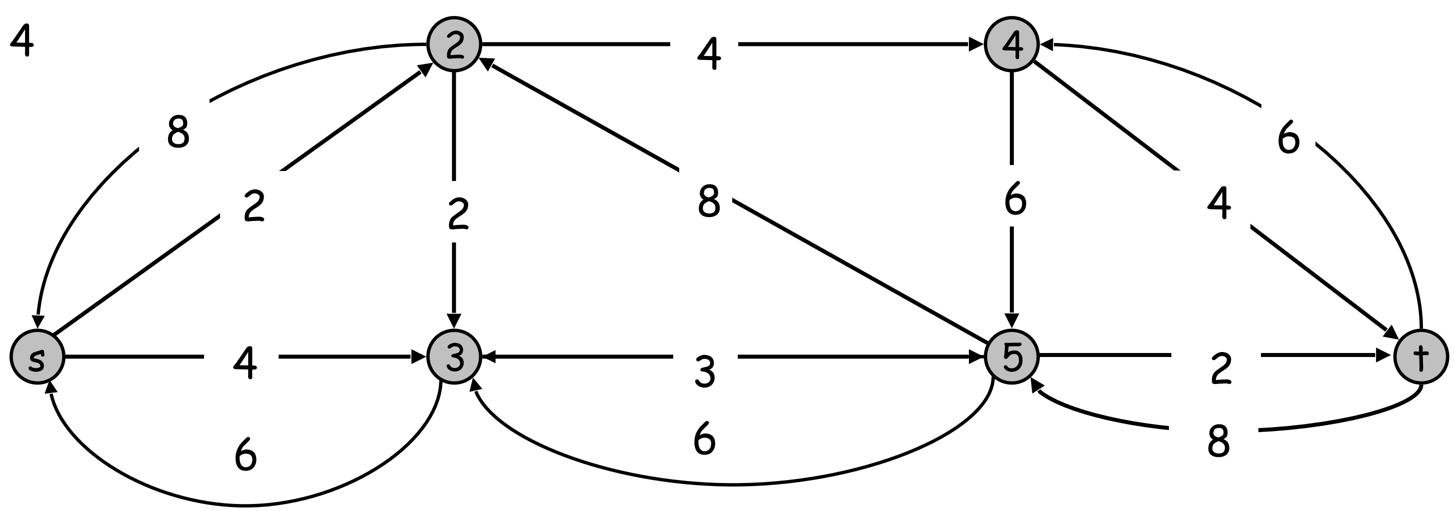
G:



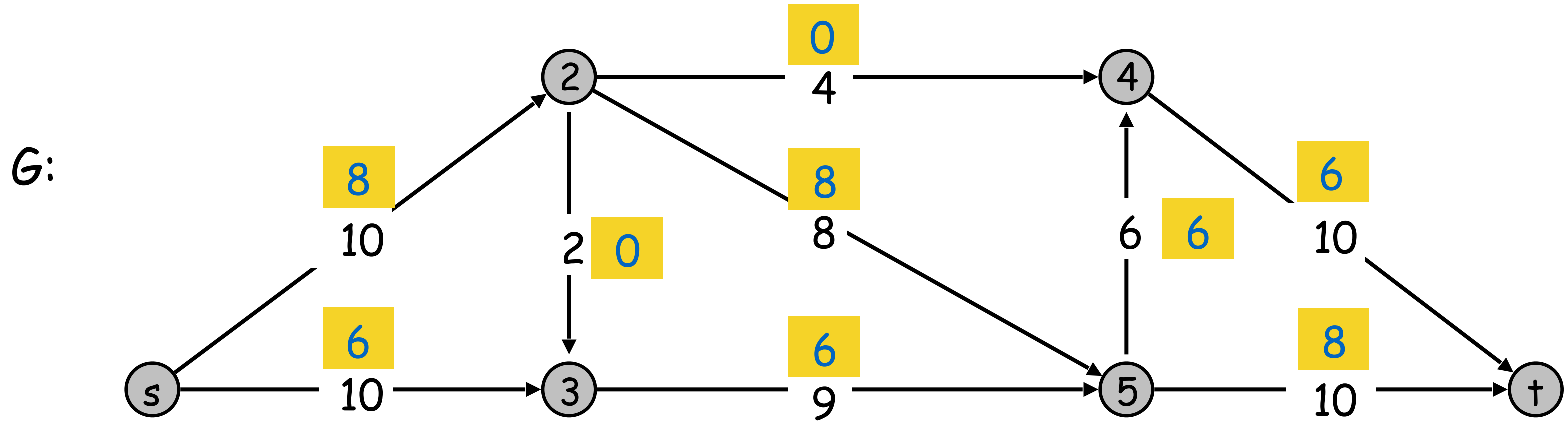

Flow value = 14

Δ = 4

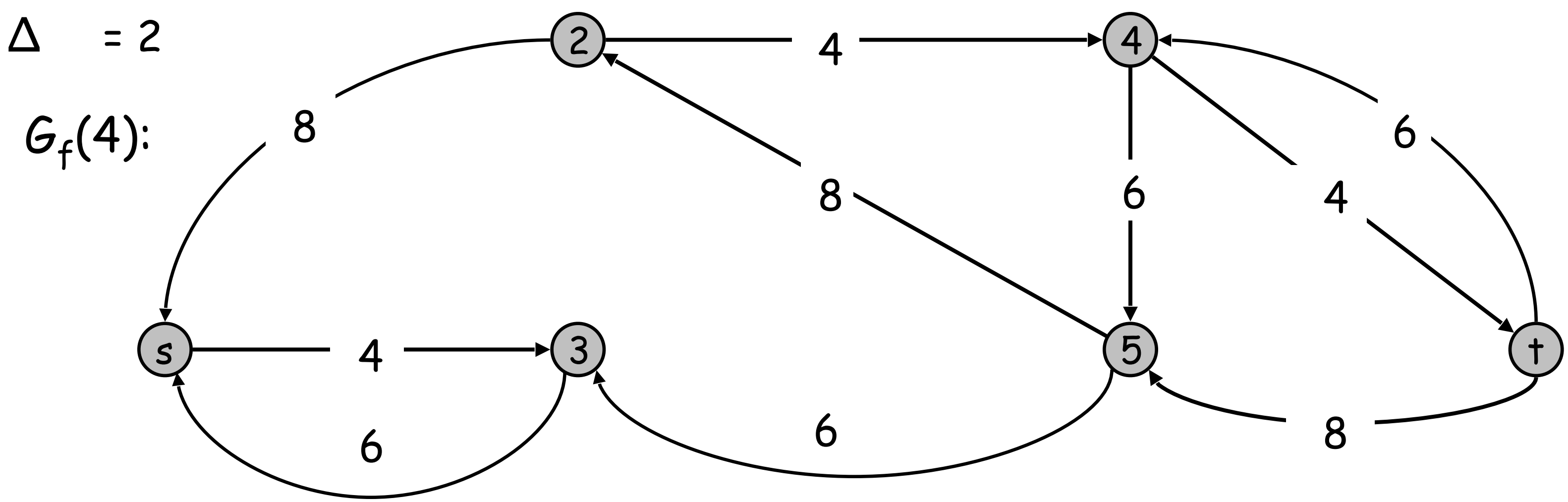
$G_f$:



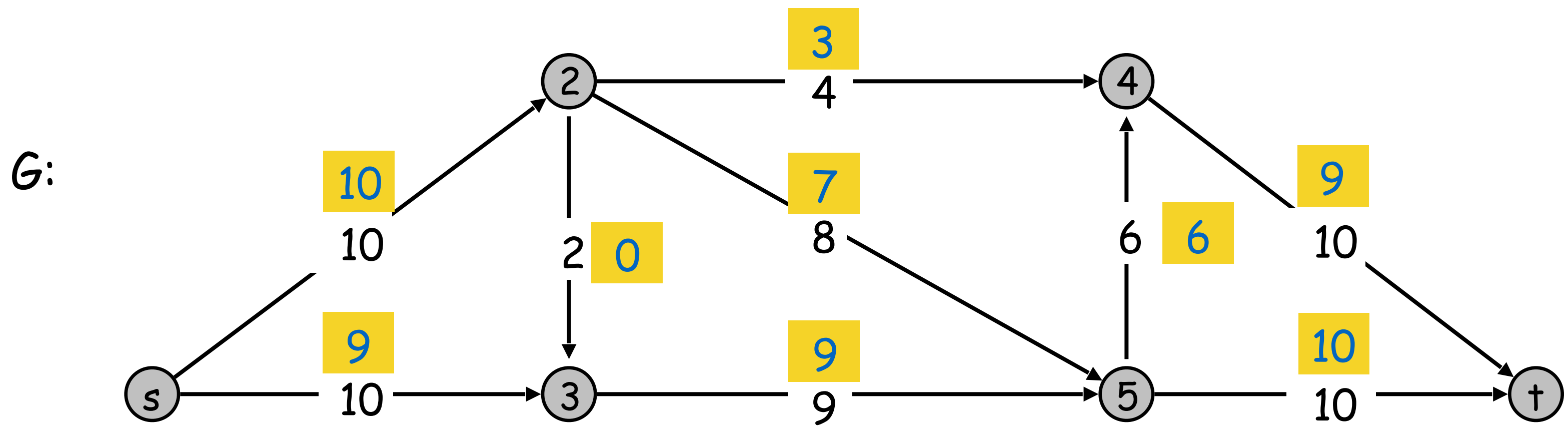
# Ford-Fulkerson Algorithm with Capacity Scaling



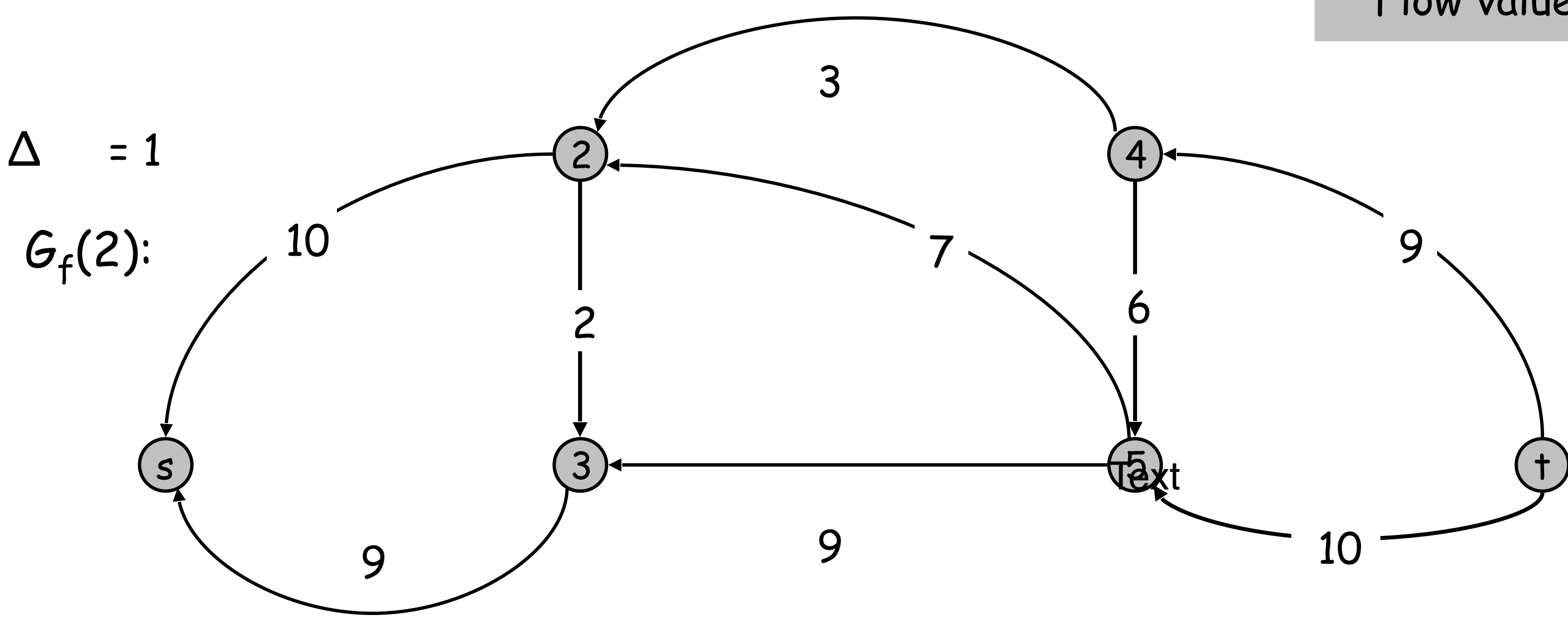

Flow value = 14



# Ford-Fulkerson Algorithm with Capacity Scaling

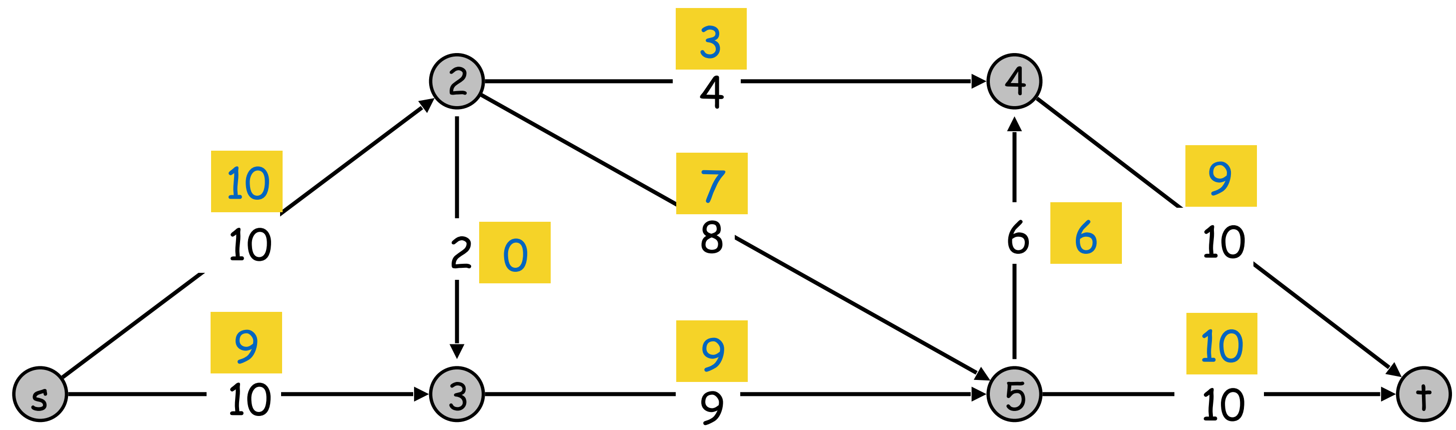



Flow value = 19



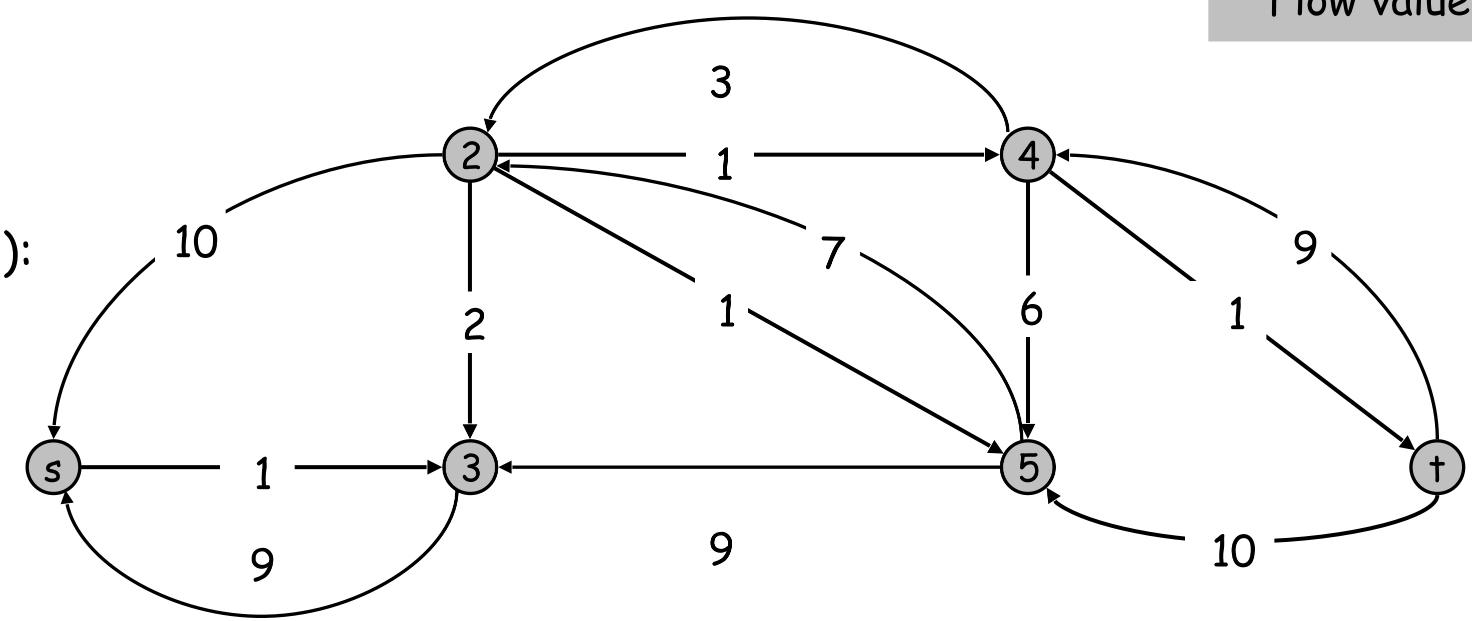
# Ford-Fulkerson Algorithm with Capacity Scaling

G:

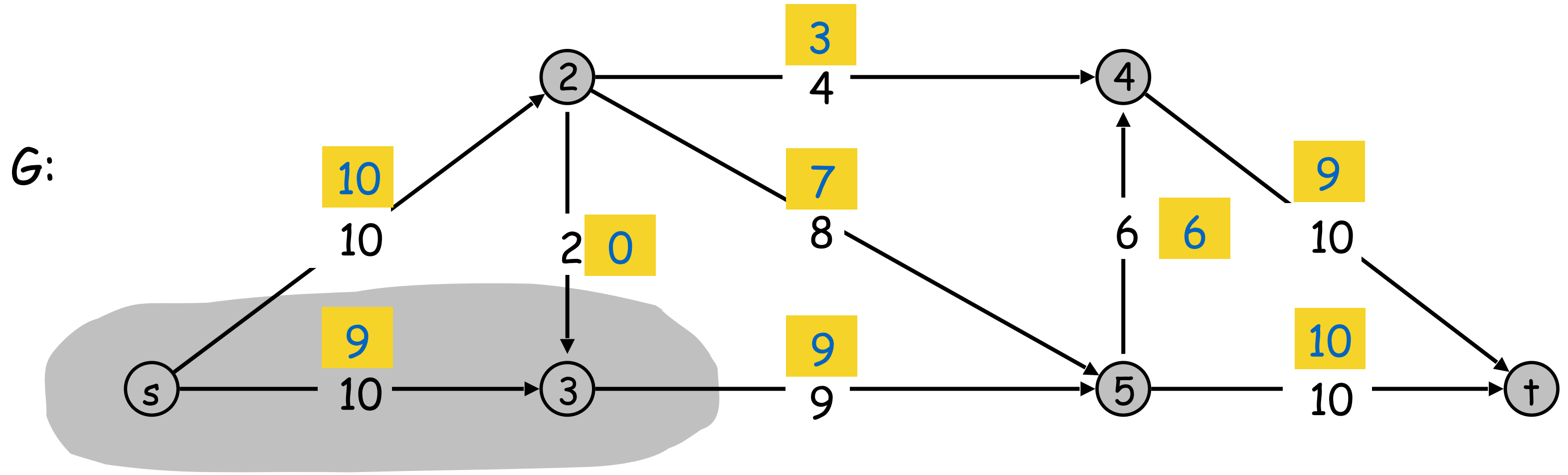



Flow value = 19

$G_f(1)$:

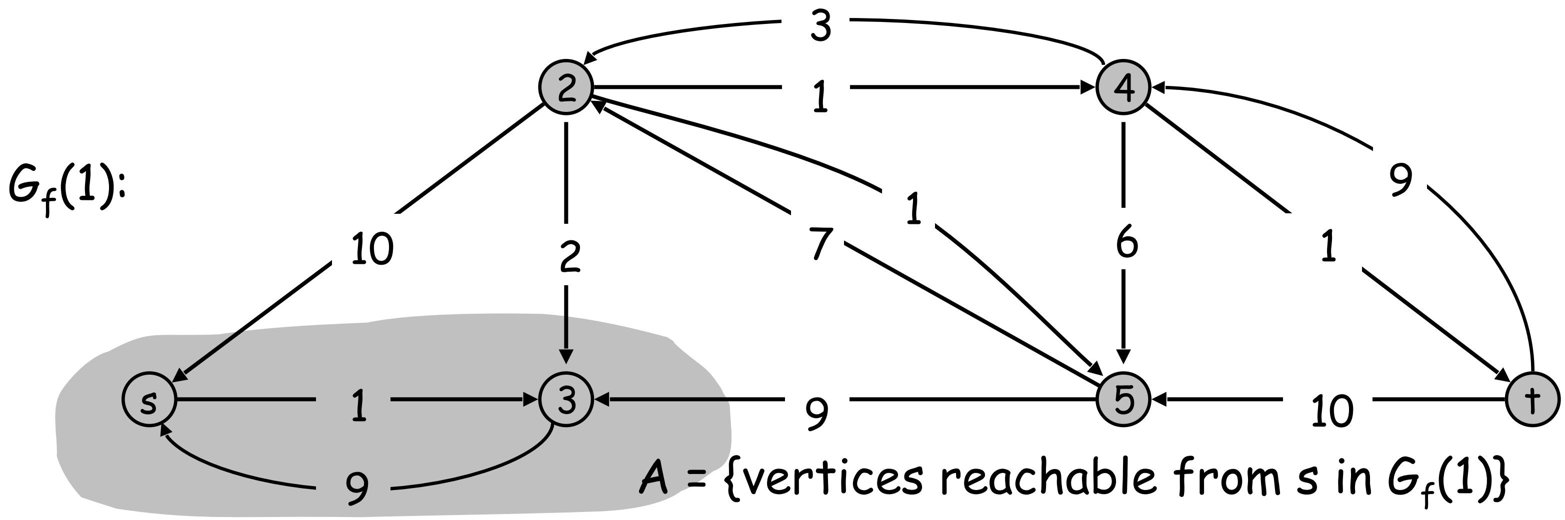


# Ford-Fulkerson Algorithm with Capacity Scaling




Cut capacity = 19

Flow value = 19




$A$ = {vertices reachable from s in $G_f(1)$}

# Capacity Scaling: Correctness

Assumption. All edge capacities are integers between 1 and $C$.

Integrality invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then f is a max flow.

Pf.

- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

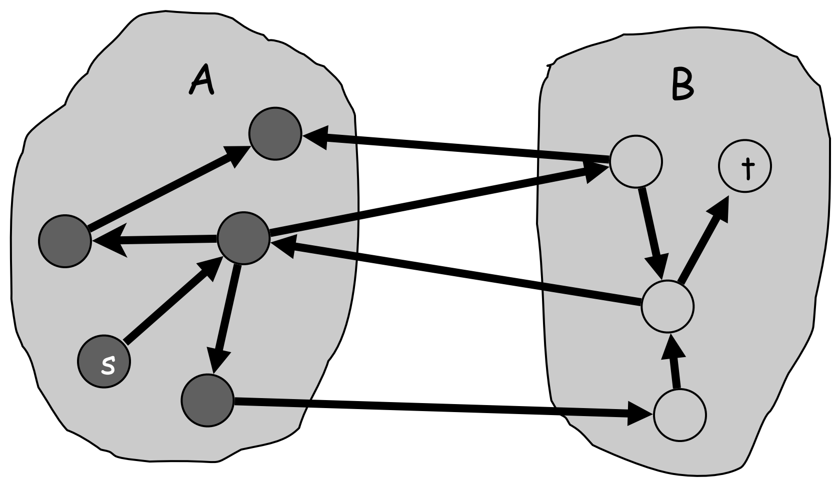
# Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow is at most v(f) + m Δ.

Pf. (almost identical to proof of max-flow min-cut theorem)

- We show that at the end of a Δ-phase, there exists a cut (A, B) such that cap(A, B) $\leq$ v(f) + m Δ.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, s $\in$ A.
- By definition of $G_f(\Delta)$, t $\notin$ A.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\ &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\ &\geq cap(A, B) - m\Delta \quad \blacksquare \end{aligned}$$




original network

# Capacity Scaling: Running Time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.
Pf. Initially $C/2 < \Delta \le C < 2\Delta$. $\Delta$ decreases by a factor of 2 each iteration. ▪

Lemma 2. Let f be the flow at the end of a $\Delta$-scaling phase. Then the value of the maximum flow is at most $v(f) + m\,\Delta$. ← proof on previous slide

Lemma 3. There are at most 2m augmentations per scaling phase.
Pf.
- Initially, each of the m edges can carry at most $C<2\Delta$ flow.
- In general, consider the situation at the beginning of a $\Delta$–phase.
- Each augmentation in a $\Delta$-phase will increase v(f) by at least $\Delta$.
- Let f be the flow at the end of the previous scaling ($2\Delta$–)phase.
- Lemma 2 $\Rightarrow$ $v(f^*) \le v(f) + m\,(2\Delta)$. ▪

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time. ▪