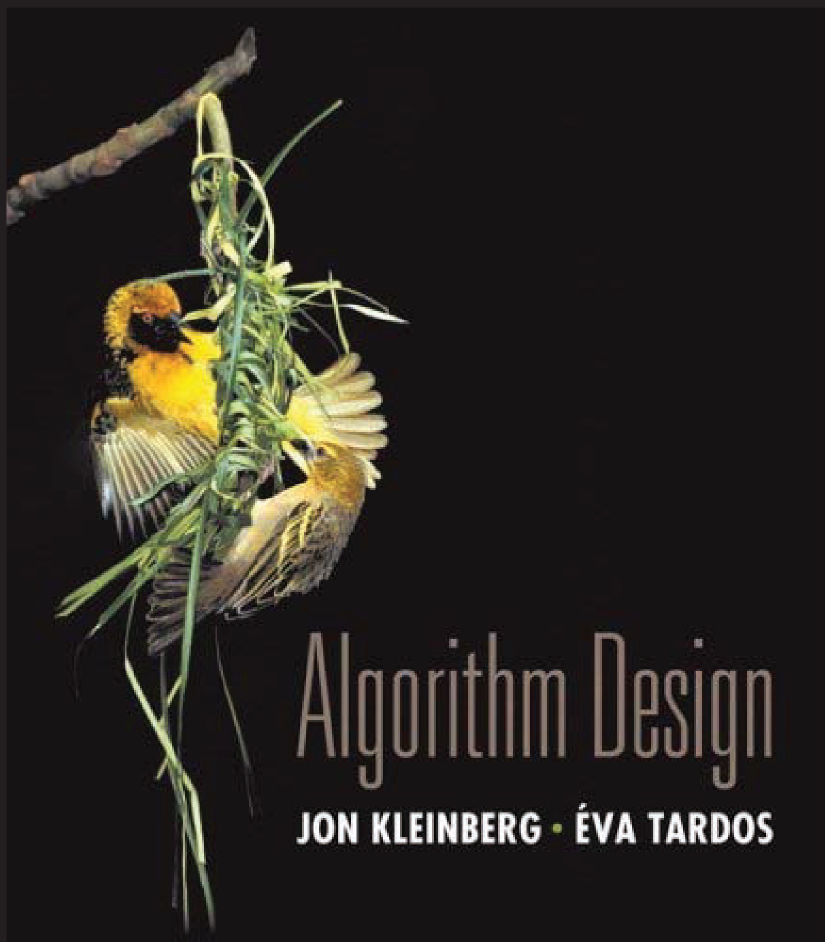
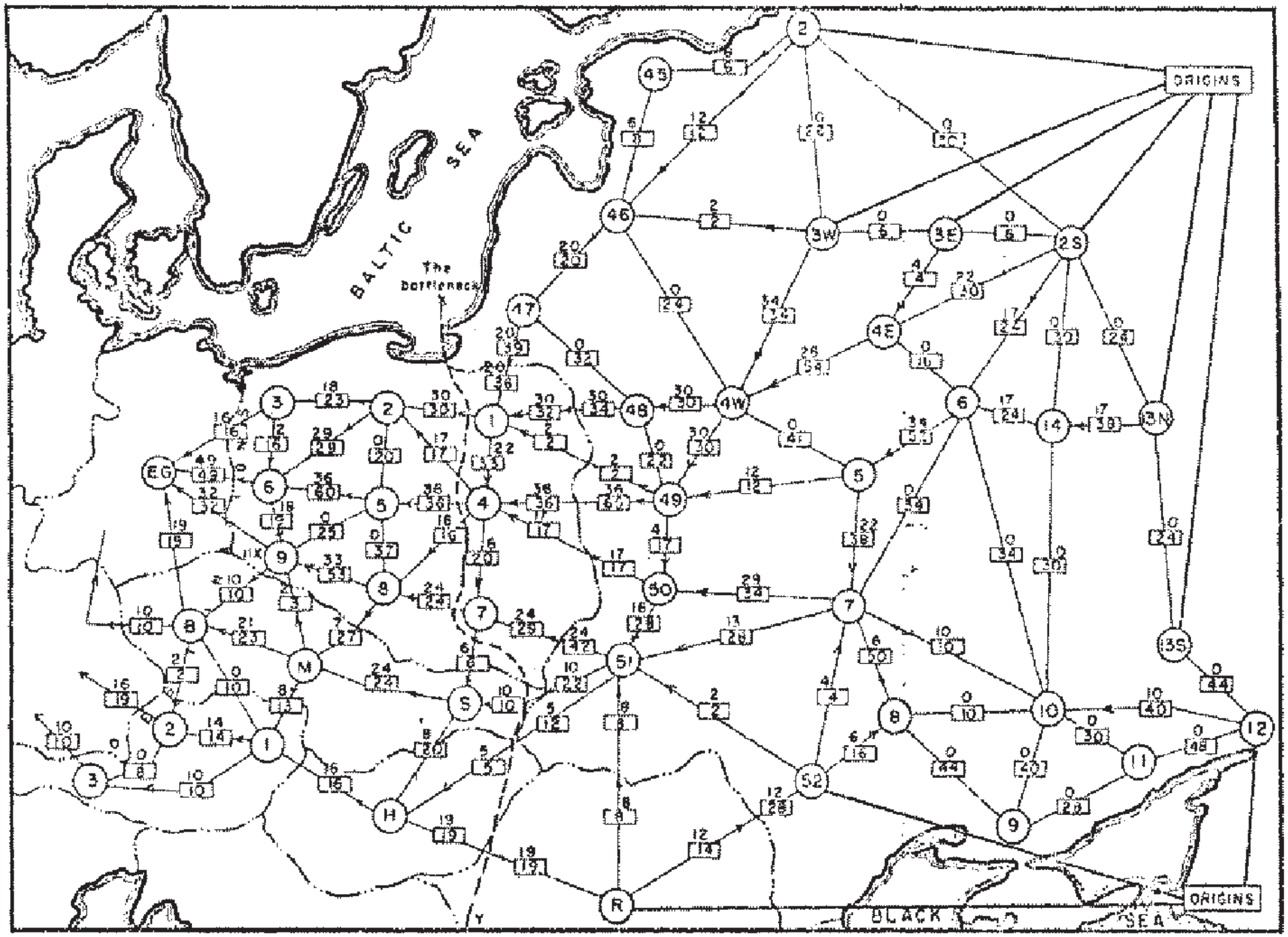


Chapter 7

Network Flow

Slides by Kevin Wayne.

Soviet Rail Network, 1955



Reference: *On the history of the transportation and maximum flow problems.*
Alexander Schrijver in Math Programming, 91: 3, 2002.

Maximum Flow and Minimum Cut

Max flow and min cut.

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

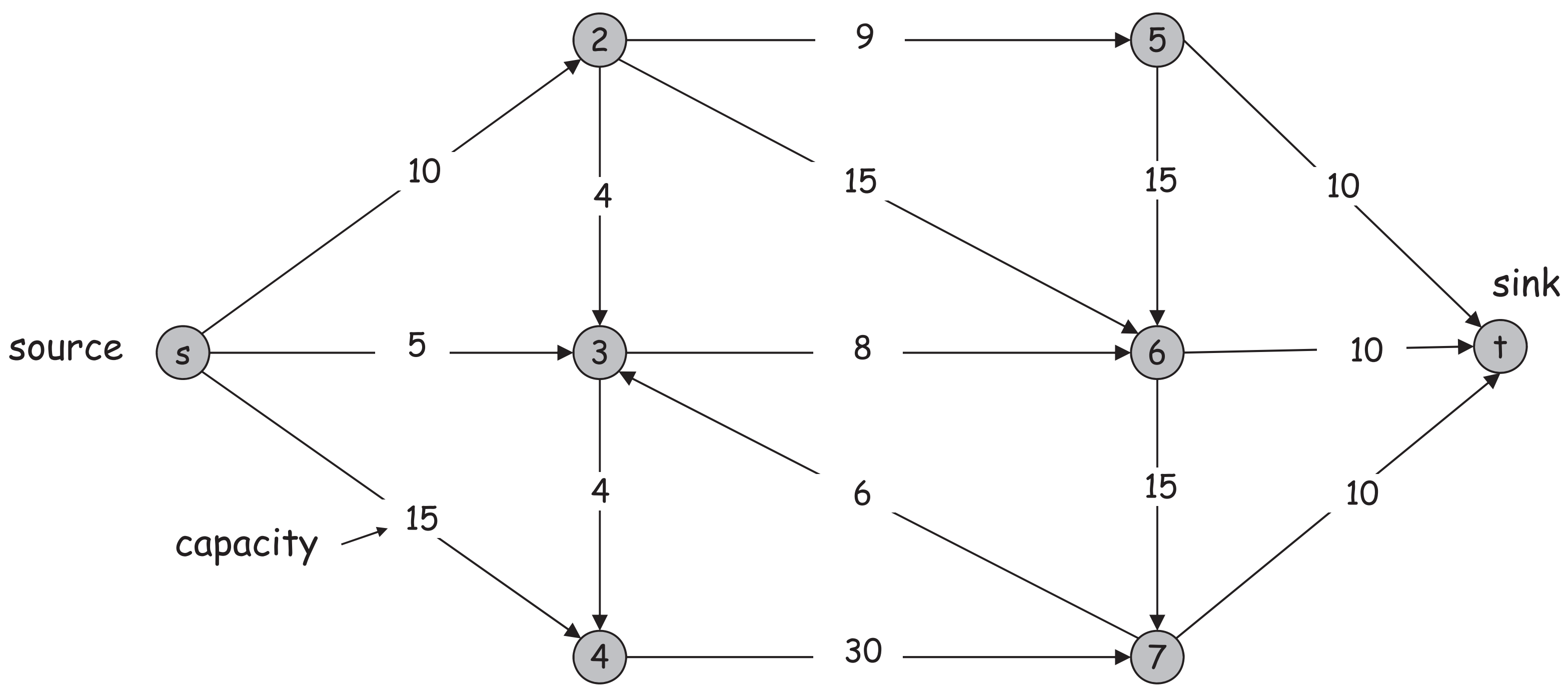
Nontrivial applications / reductions.

- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more …

Minimum Cut Problem

Flow network.

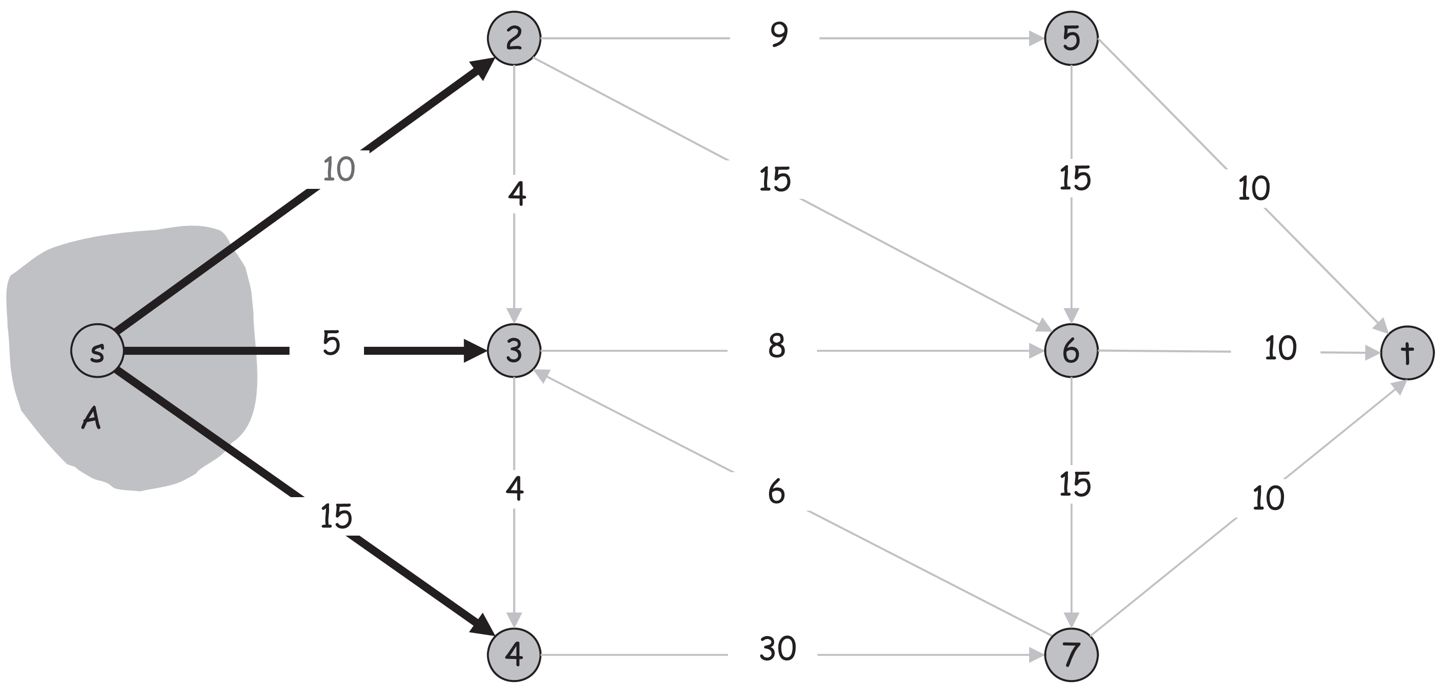
- Abstraction for material flowing through the edges.
- $G = (V, E)$ = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- $c(e)$ = capacity of edge e.



Cuts

Def. An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.

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

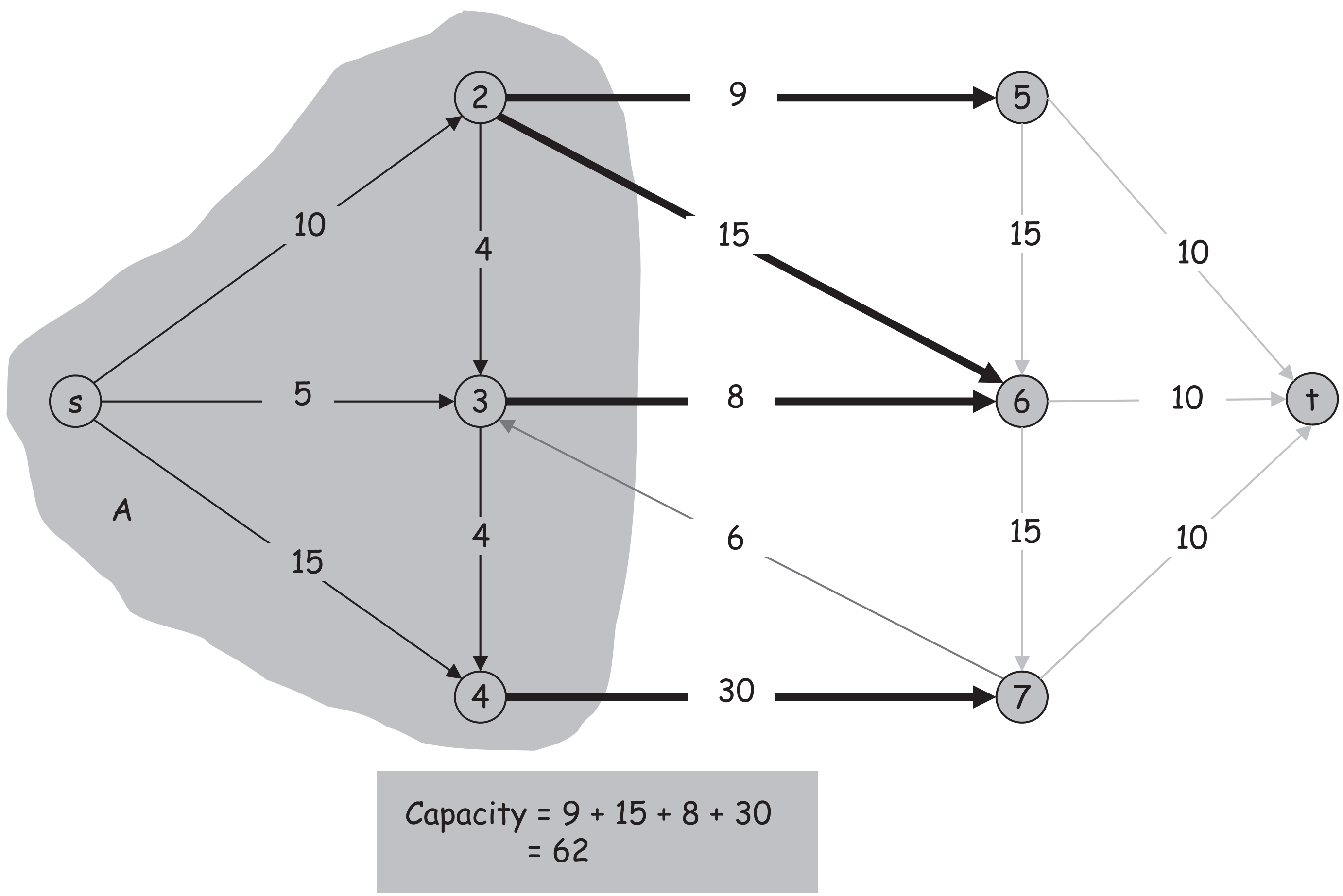


Capacity = 10 + 5 + 15
= 30

Cuts

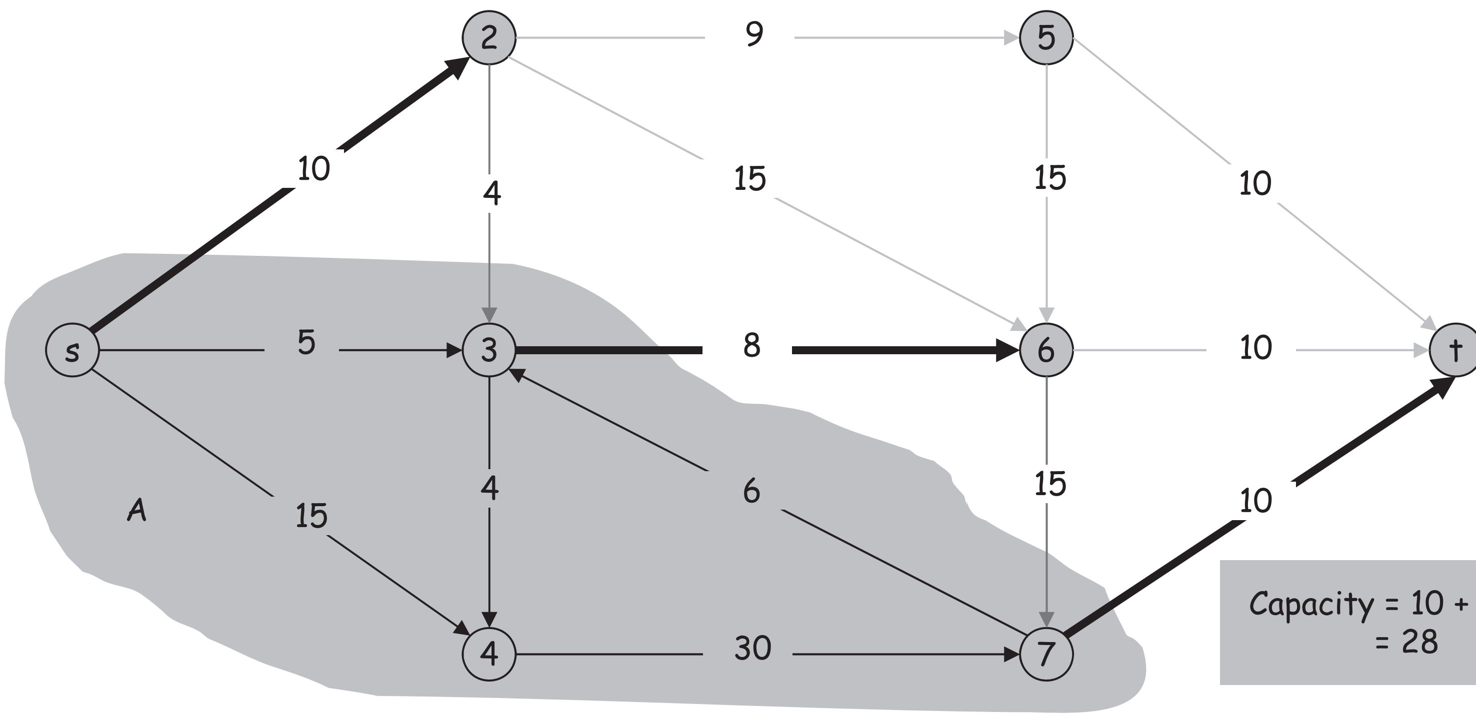
Def. An s-t cut is a partition (A, B) of V with s $\in$ A and t $\in$ B.

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



Minimum Cut Problem

Min s-t cut problem. Find an s-t cut of minimum capacity.



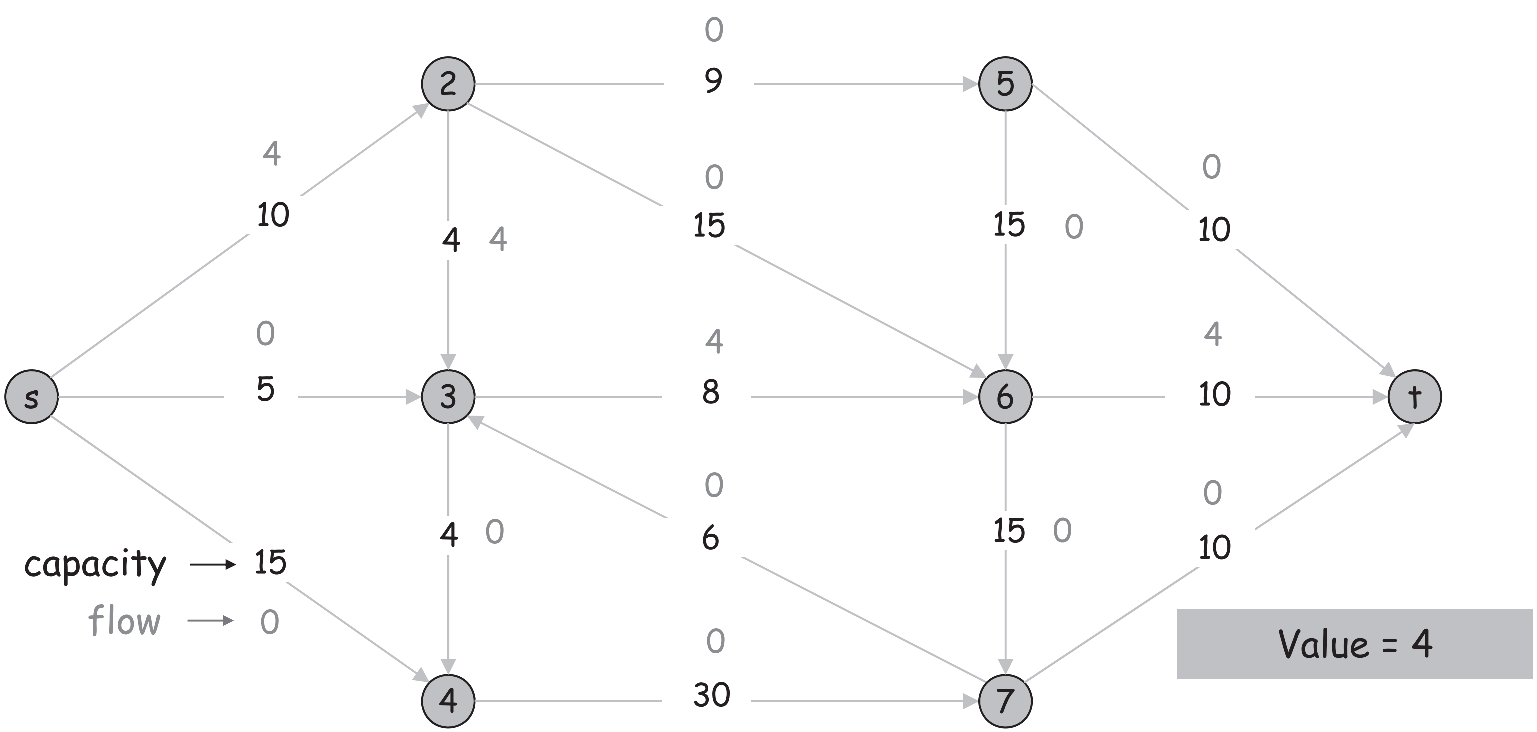
Capacity = 10 + 8 + 10
= 28

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

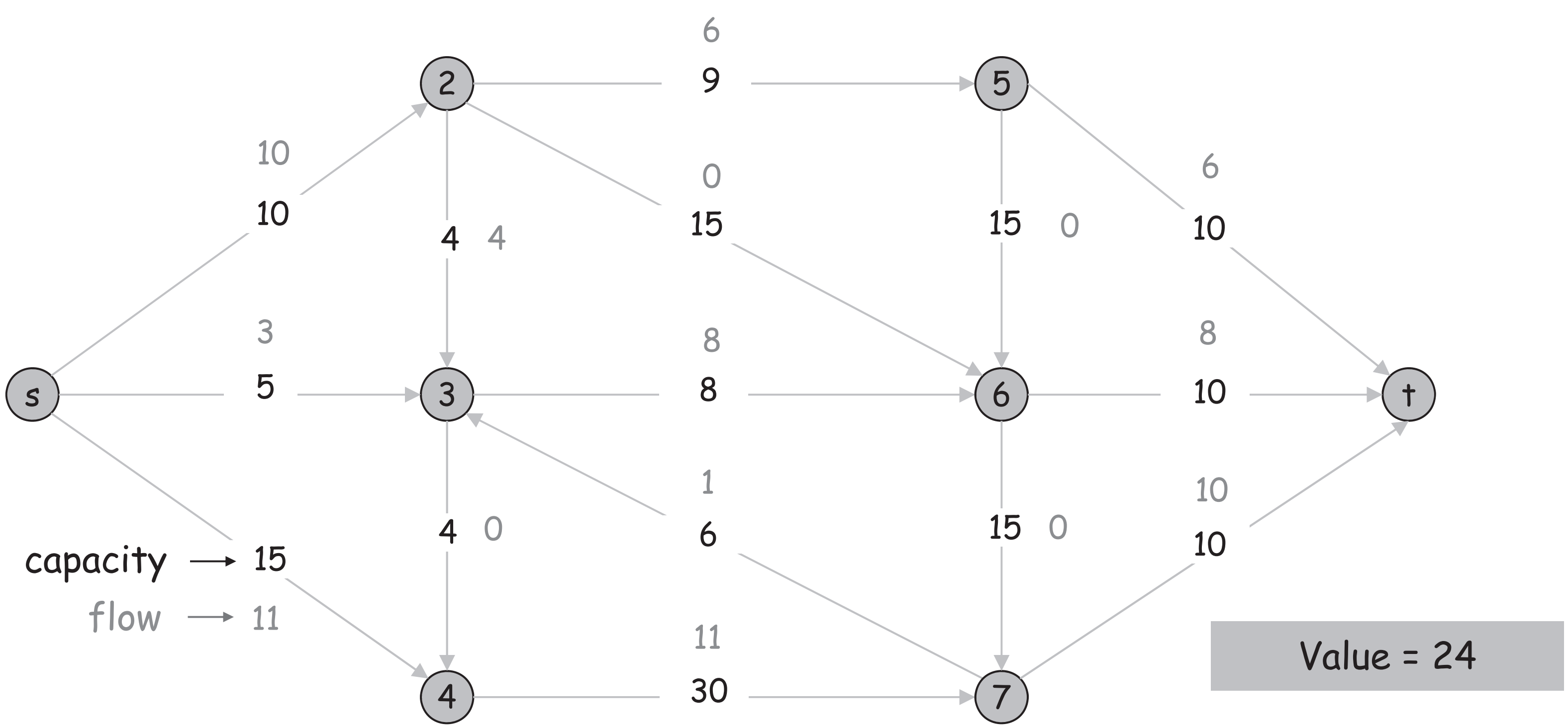


Flows

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

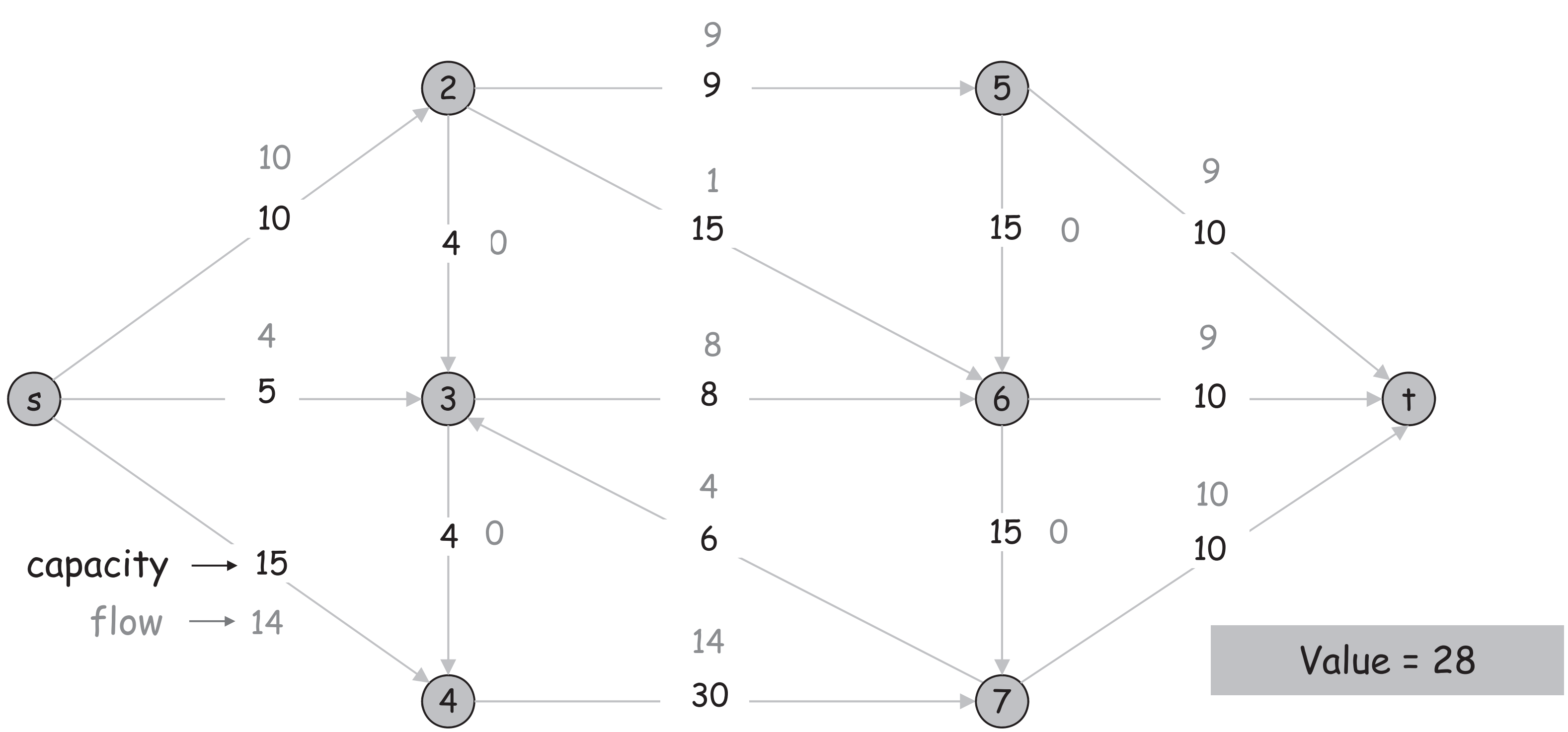
- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

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



Maximum Flow Problem

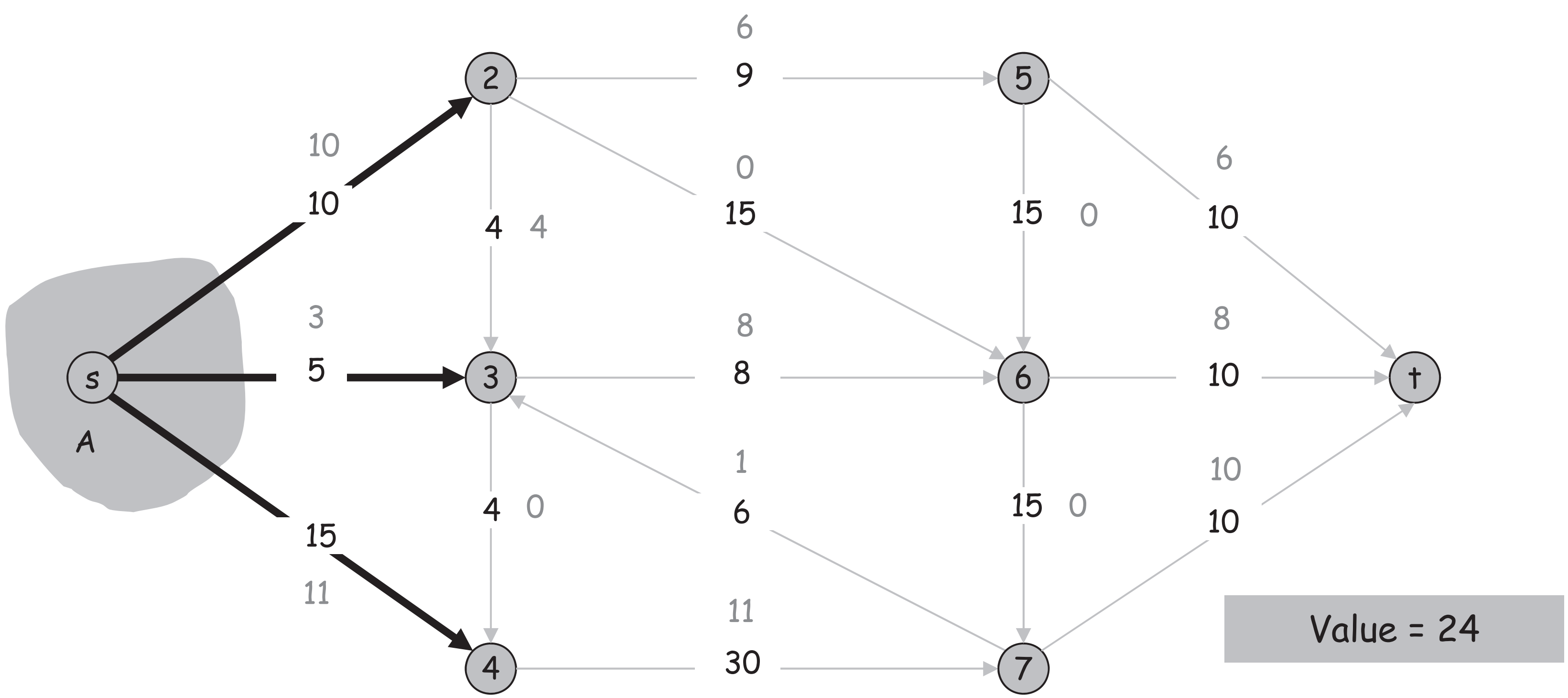
Max flow problem. Find s-t flow of maximum value.



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.

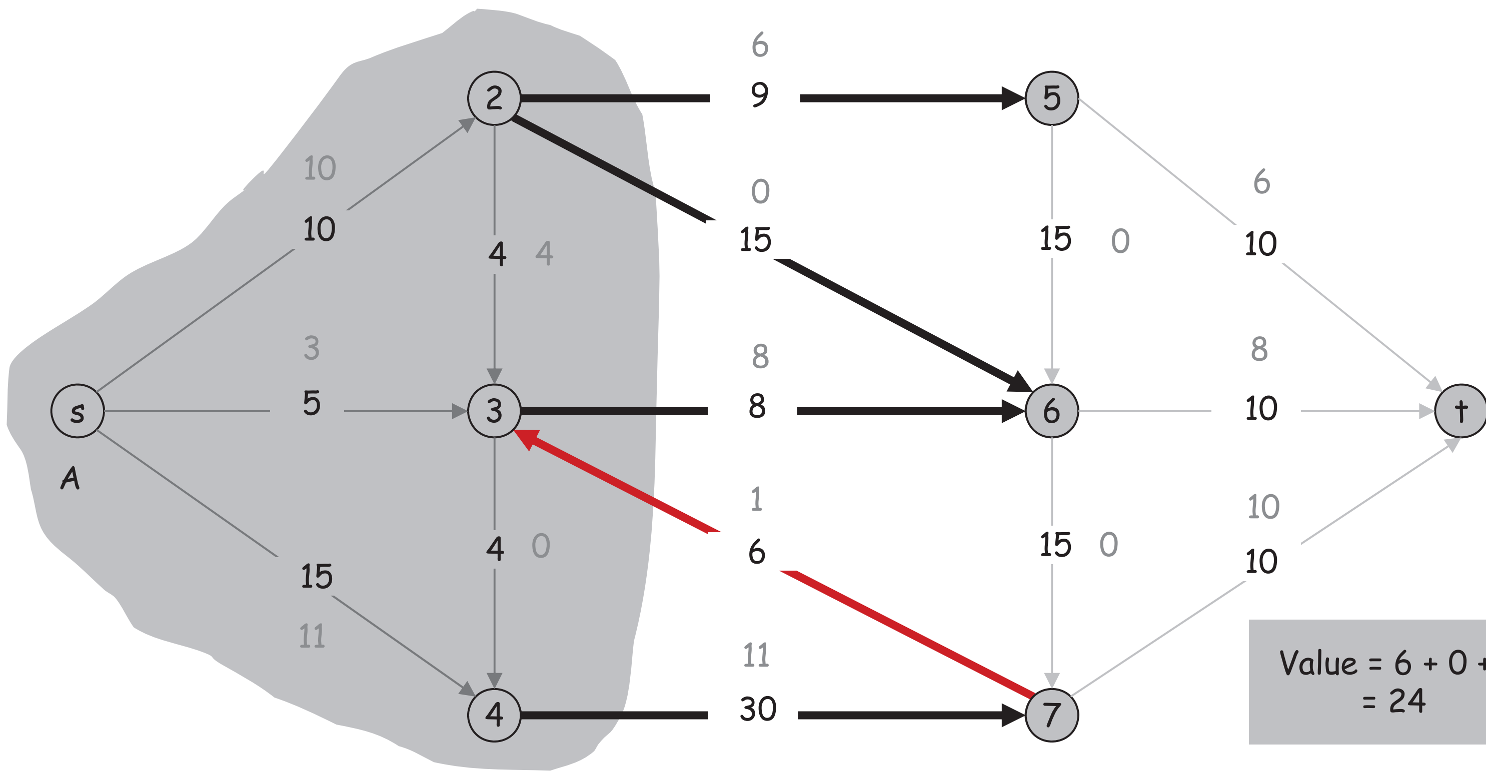
$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



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.

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

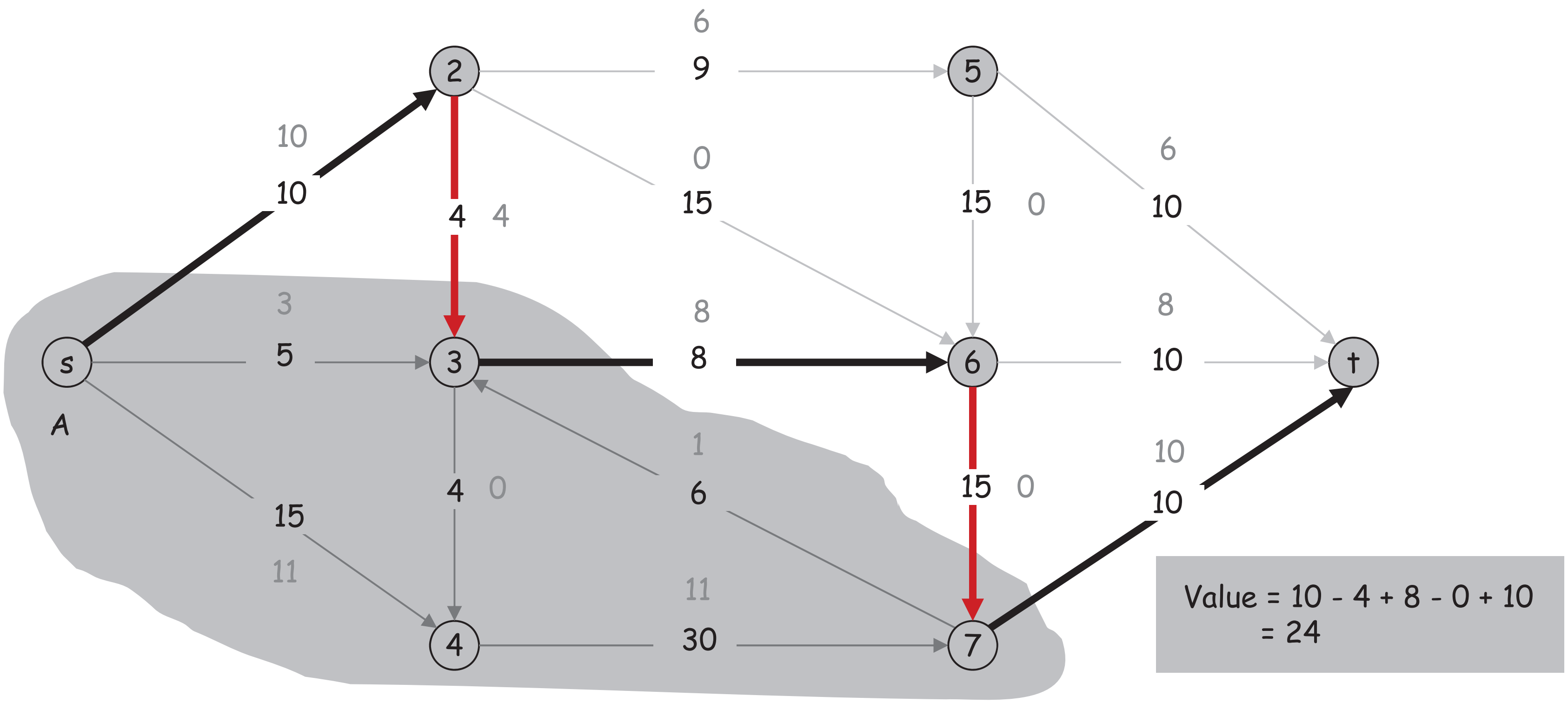


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

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



Flows and Cuts

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

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

Proof?

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 $\longrightarrow$

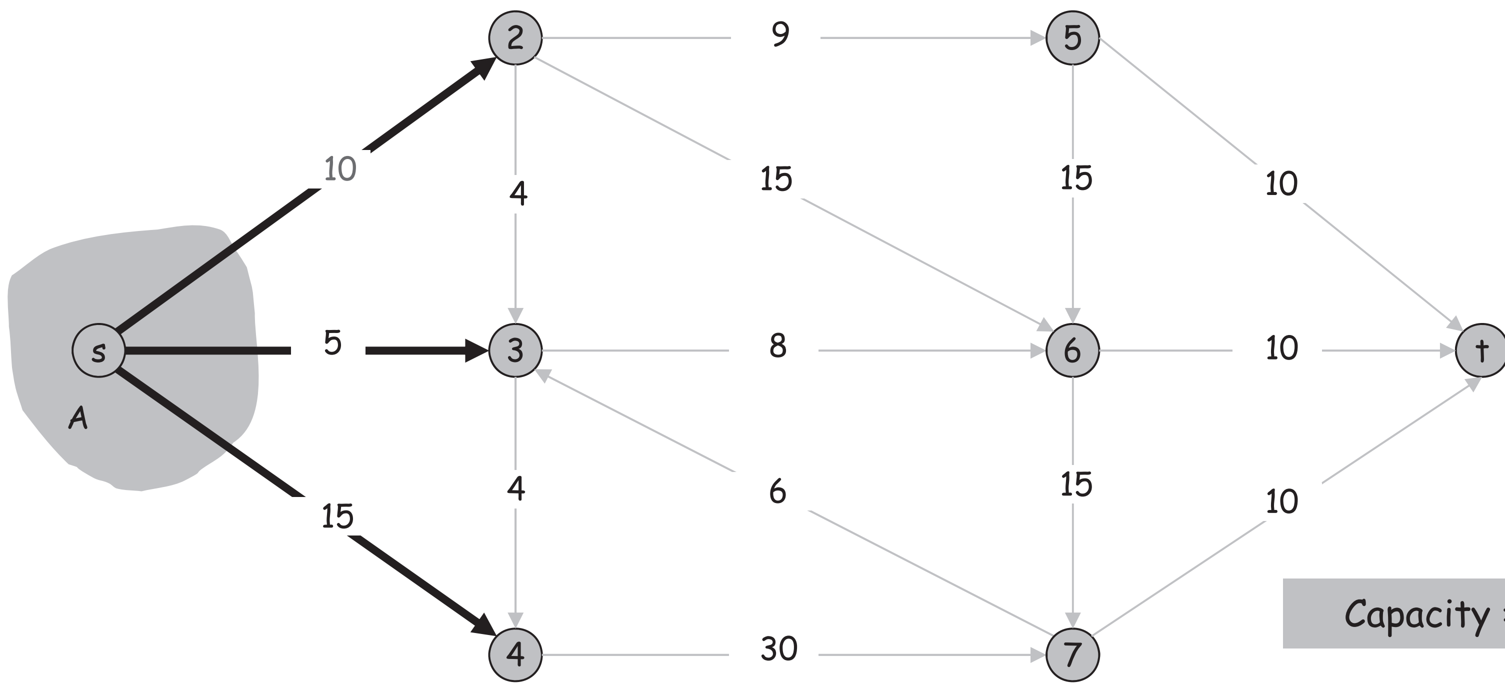
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

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

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

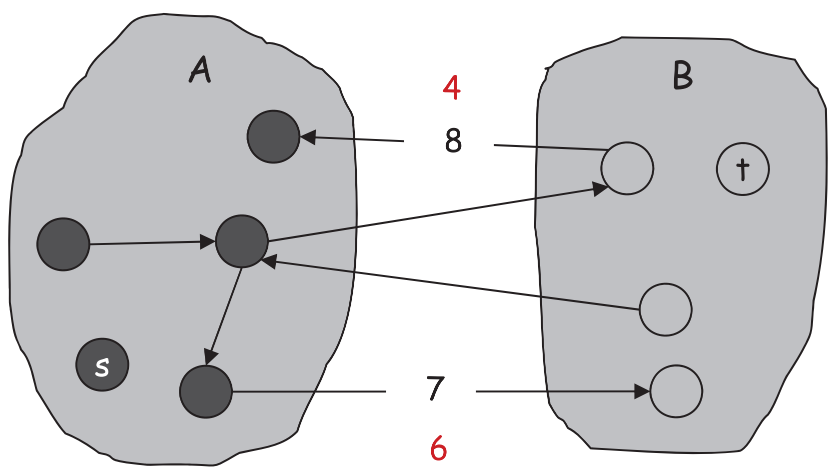
Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \leq cap(A, B)$.

Proof?.

Flows and Cuts

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \leq \text{cap}(A, B)$.

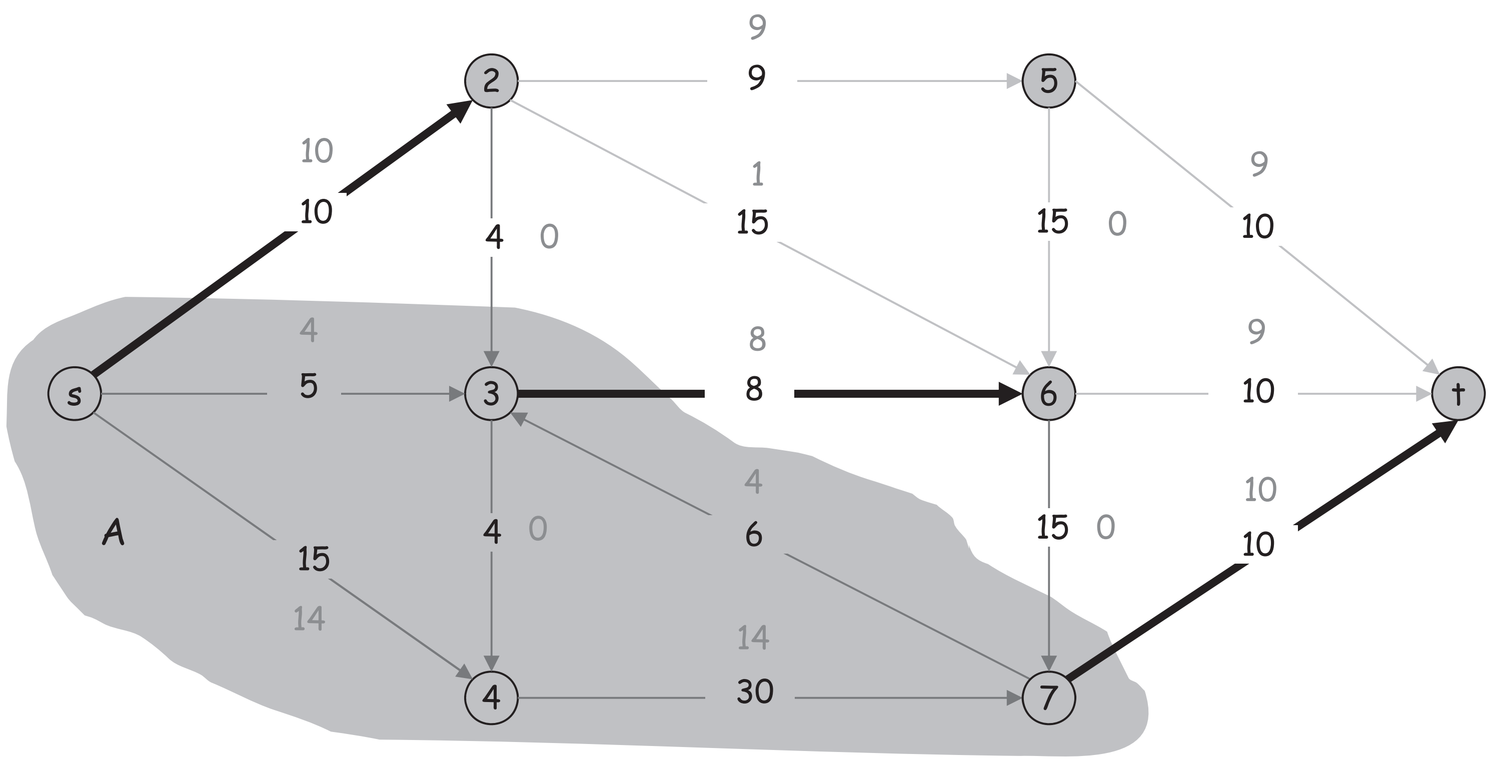
Pf.
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\leq \sum_{e \text{ out of } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \end{aligned}$$
▪



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.

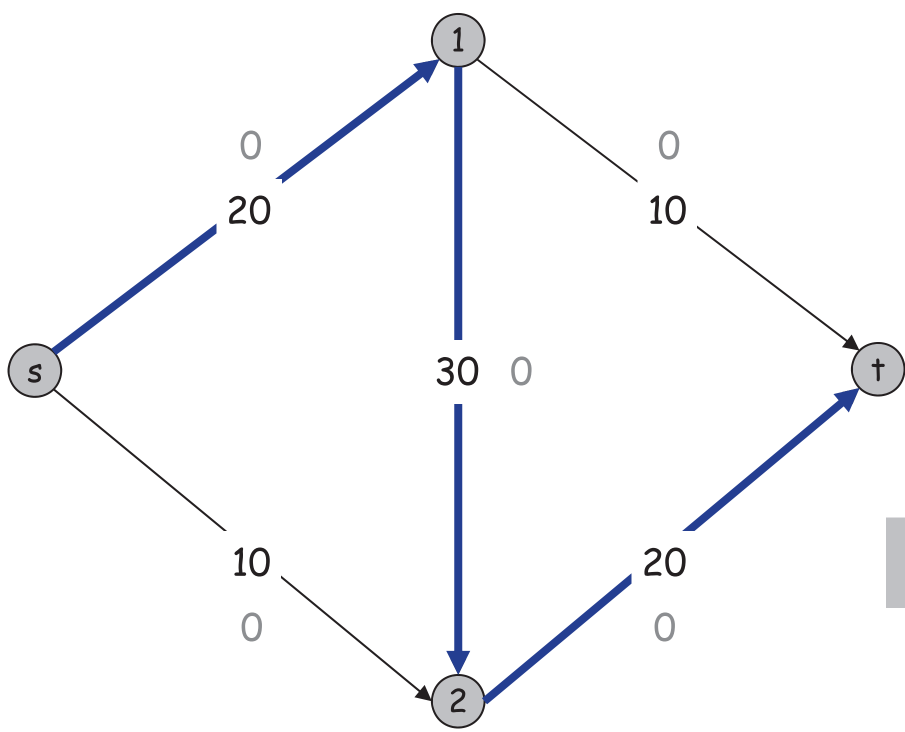
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



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.

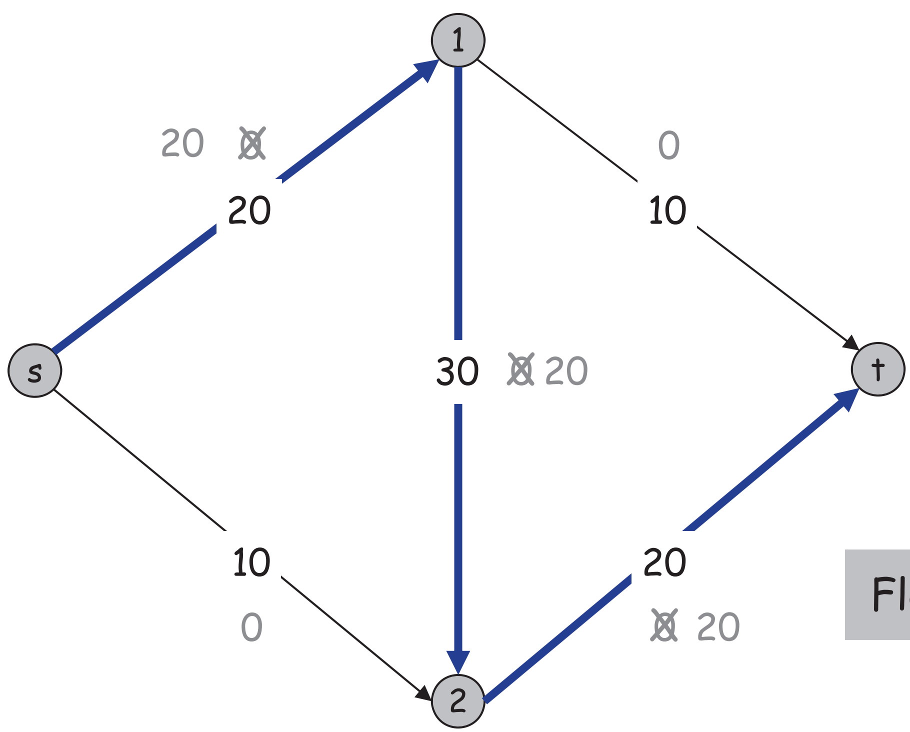


Flow value = 0

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.



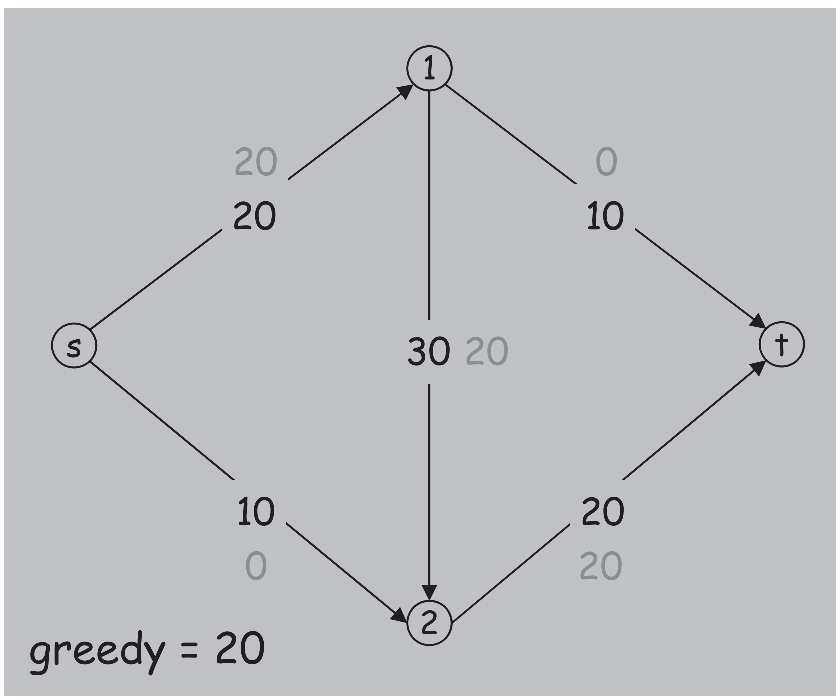
Flow value = 20

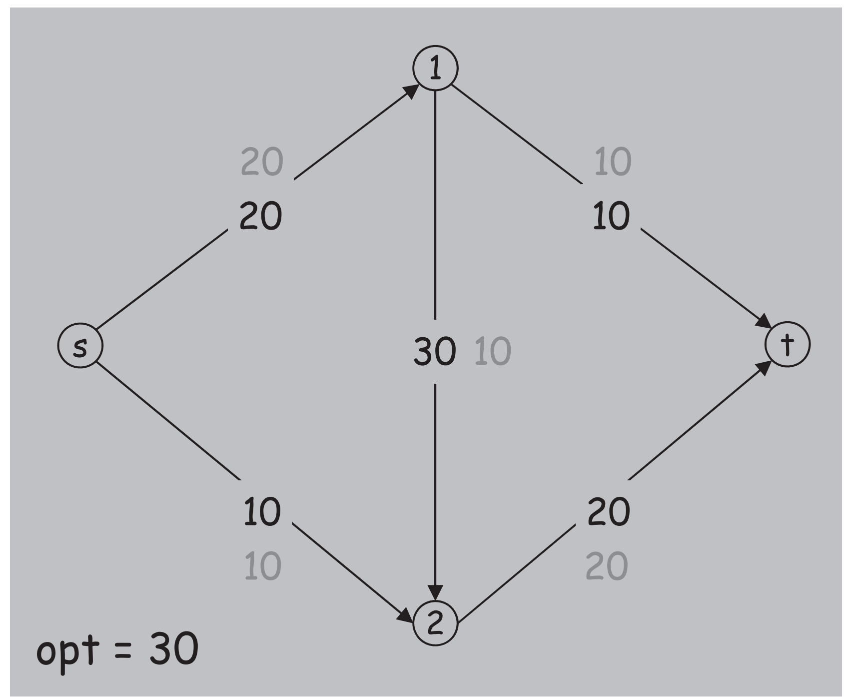
Towards a Max Flow Algorithm

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.

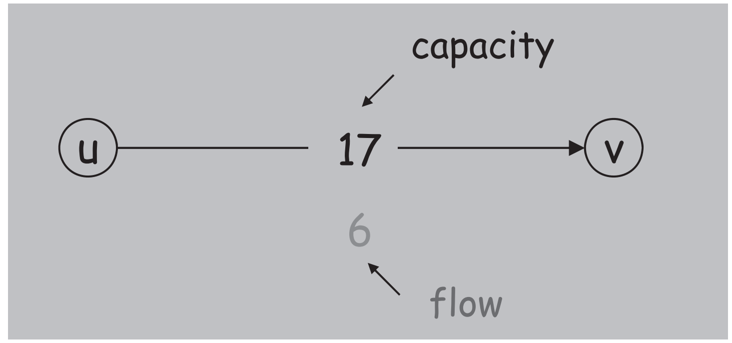
locally optimality $\not\Rightarrow$ global optimality





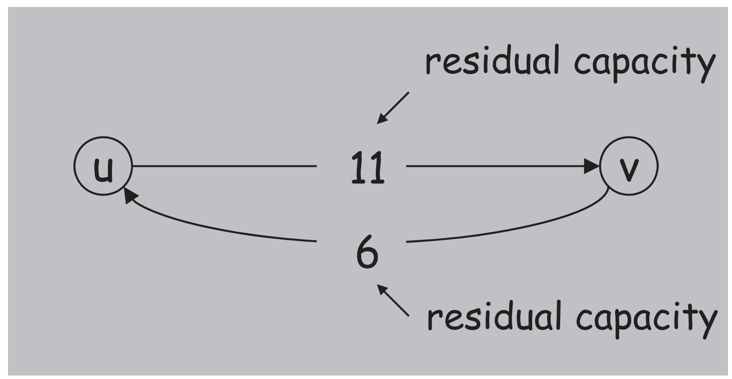
Residual Graph

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:

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



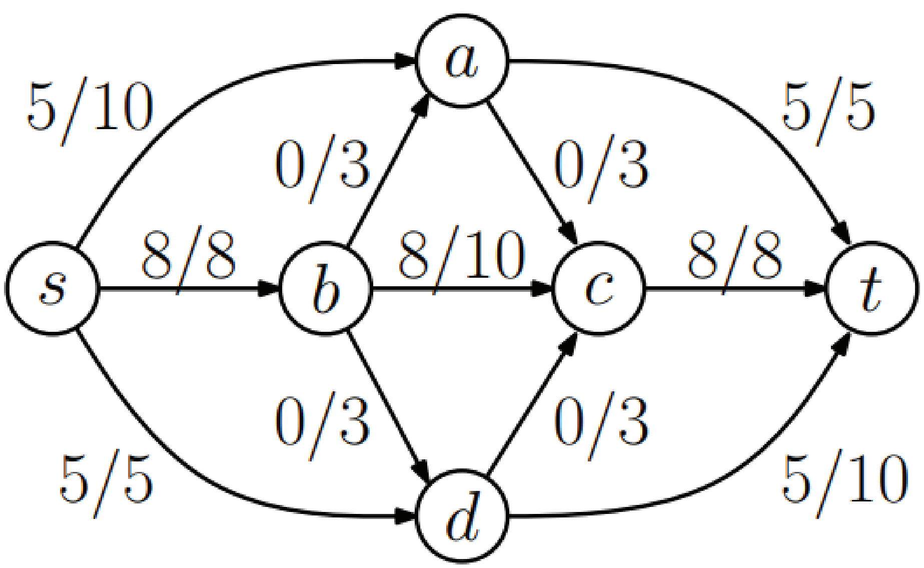
Residual graph: $G_f = (V, E_f)$.
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Consider a network G. Let

- *f* be a flow in G, and *Gf* the associated residual network.
  Recall for each edge *(u,v)* where

- ***f(u,v) < c(u,v)***, create edge *(u,v)* in *Gf* with capacity *cf(u,v)=c(u,v)-f(u,v)*.
  [*i.e. we can increase flow along this edge by up to cf(u,v) units*]

- ***f(u,v)>0***, create edge *(v,u)* in *Gf* with capacity *cf(v,u)=f(u,v)*. [*i.e. we can decrease the existing flow by up to cf(u,v) units*]



(a): A flow f in network G

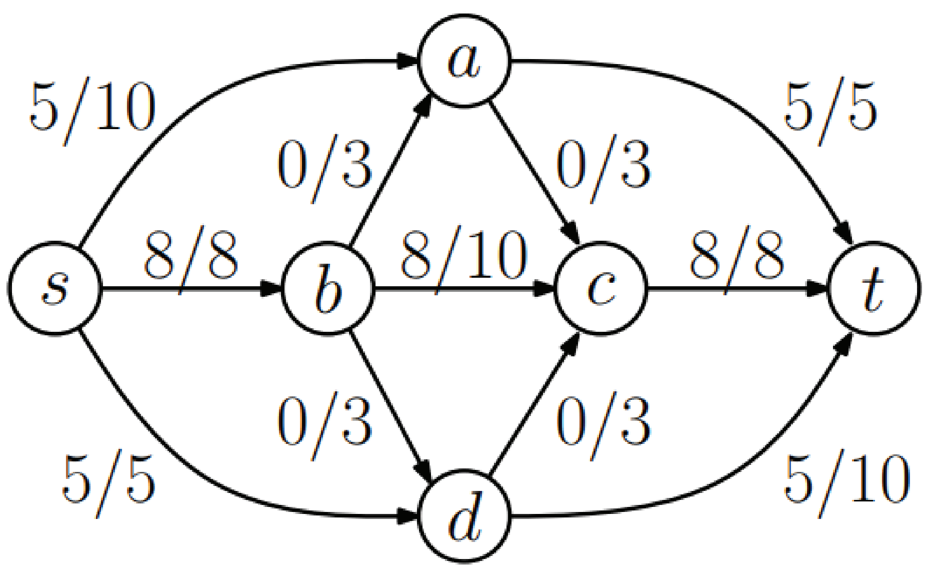
Residual network?

Consider a network G. Let

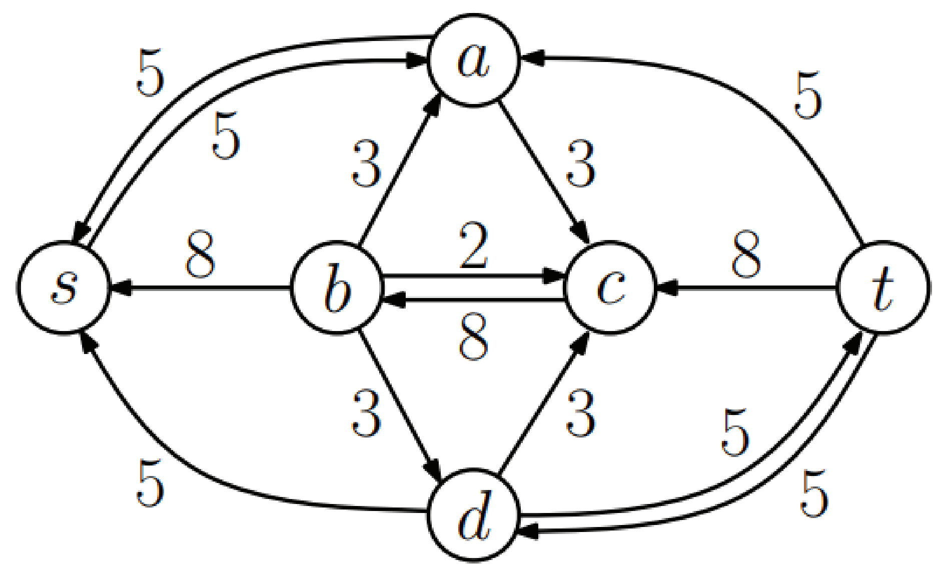
- *f* be a flow in G, and *Gf* the associated residual network.
  Recall for each edge *(u,v)* where

- ***f(u,v) < c(u,v)***, create edge *(u,v)* in *Gf* with capacity *cf(u,v)=c(u,v)-f(u,v)*.
  [i.e. we can increase flow along this edge by up to cf(u,v) units]

- ***f(u,v)>0***, create edge *(v,u)* in *Gf* with capacity *cf(v,u)=f(u,v)*. *[i.e. we can decrease the existing flow by up to cf(u,v) units]*



(a): A flow f in network G

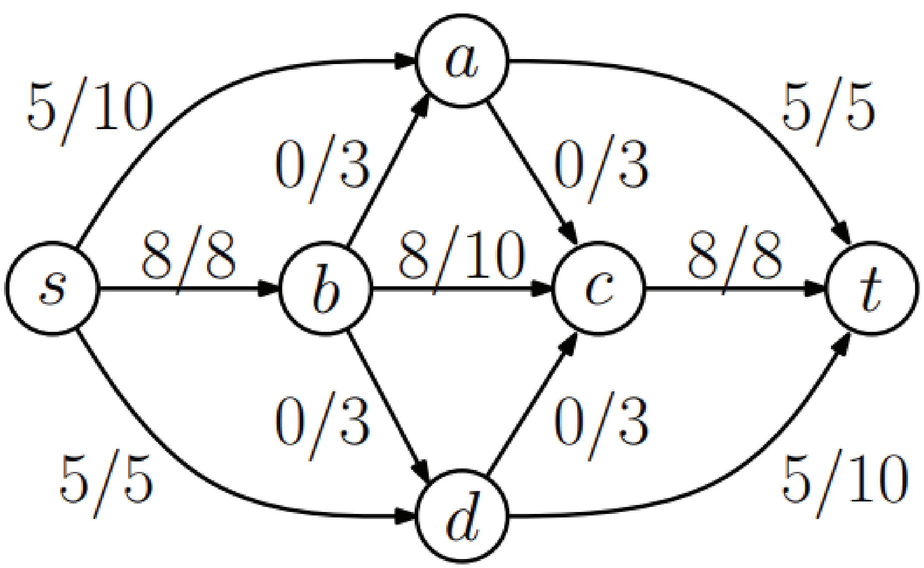


(b): Residual network G_f

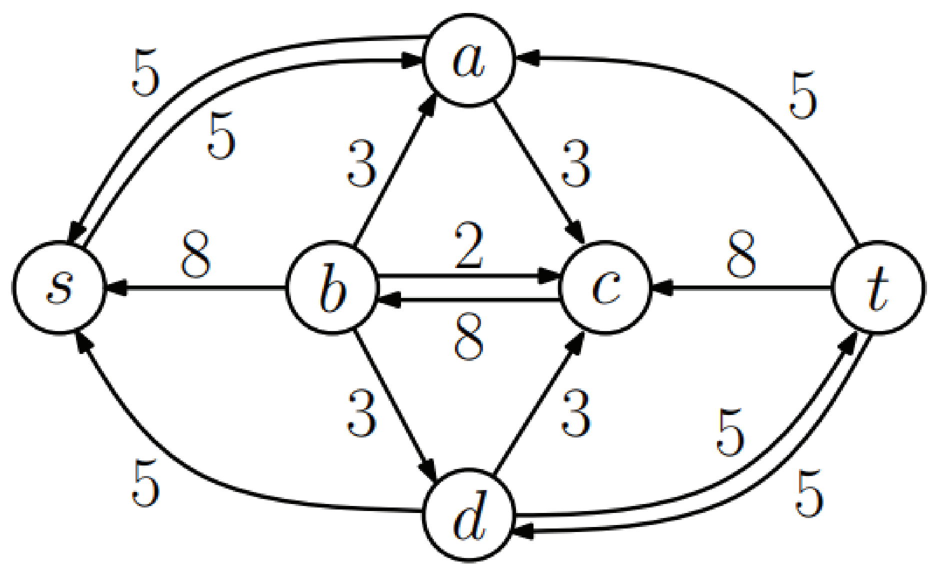
An ***augmenting path*** is a simple path P from s to t in Gf .

The ***residual capacity*** (also called the bottleneck capacity) ***of the path*** is the minimum capacity of any edge on the path. Denote this by $cf(P)$.

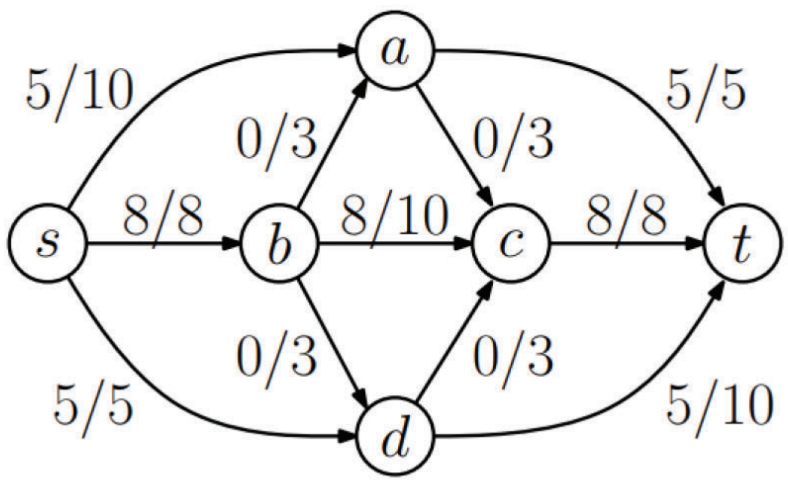
Exercise: Find augmenting path in the following network.



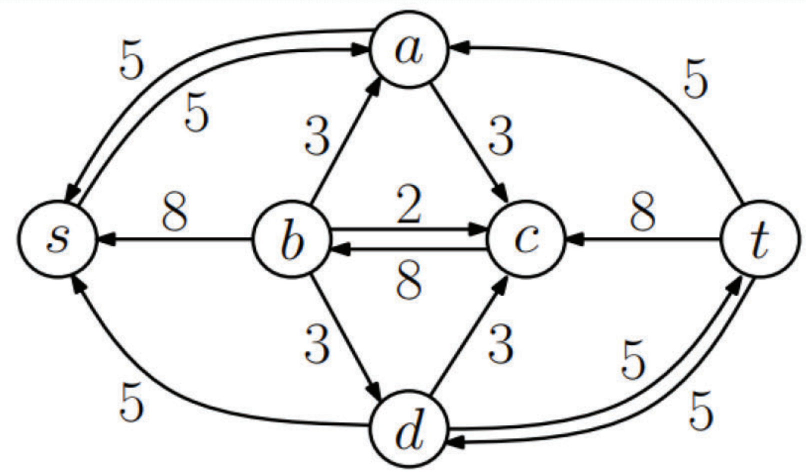
(a): A flow f in network G



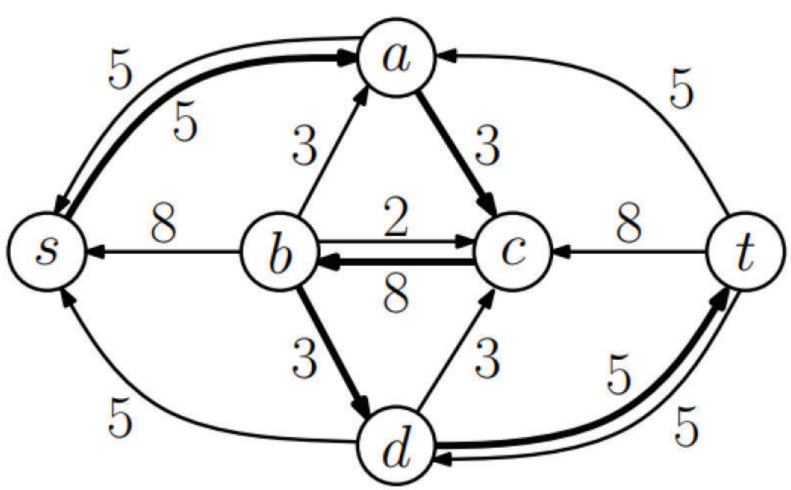
(b): Residual network G_f



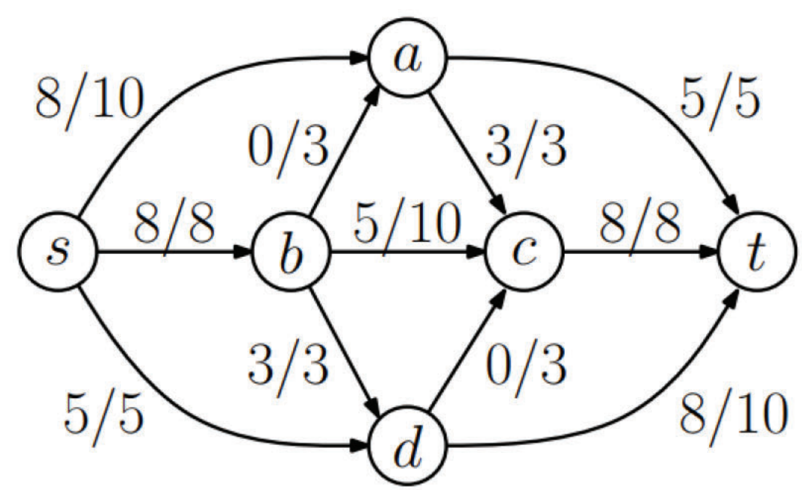
(a): A flow f in network G



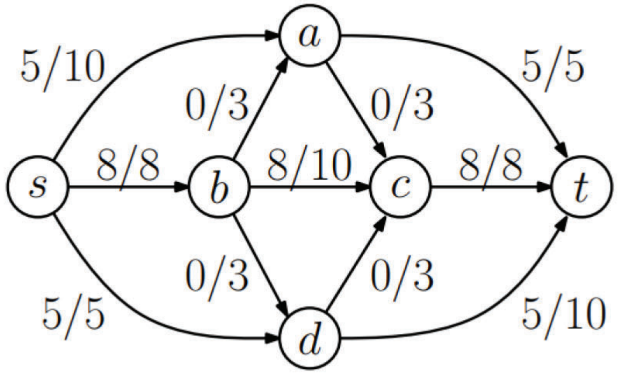
(b): Residual network G_f



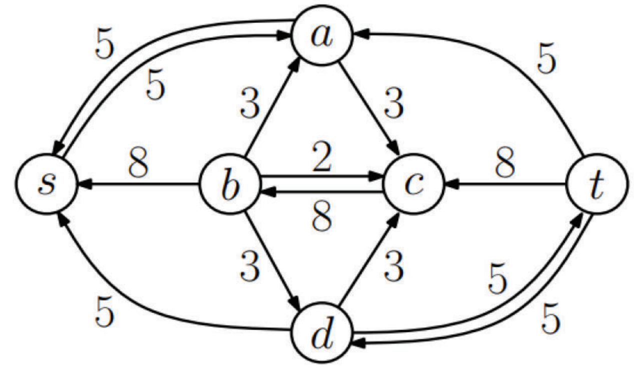
(c) Augmenting path of capacity 3



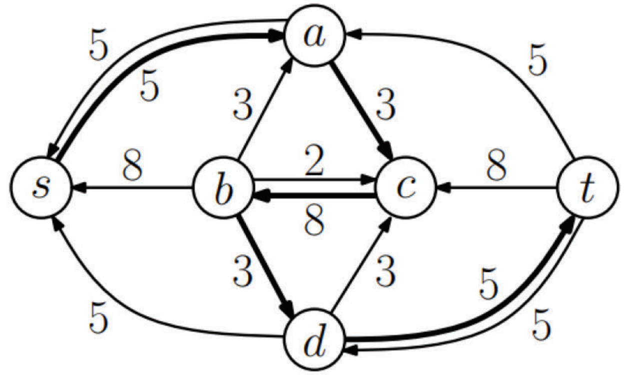
(d) The flow after augmentation



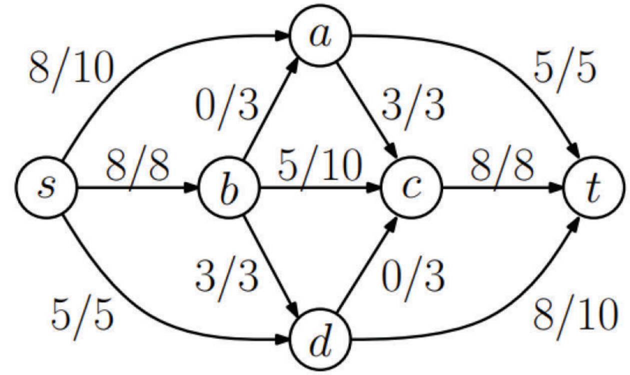
(a): A flow f in network G



(b): Residual network G_f



(c) Augmenting path of capacity 3



(d) The flow after augmentation

Difference from Greedy:

- The greedy algorithm only increases flow on the edges
- An augmenting path may increase flow on a back edge, thus decreasing flow on some edge of the initial network G.

Capacity of each edge in the residual network := its residual capacity.

Observation (informal): if we can push flow through the residual network then we can push this additional amount of flow through the original network.

Sum of flows: Given two flows f and f', define their **sum** $f'' = f + f'$ by:

- $f''(u,v)=f(u,v) + f'(u,v)$.

Notation. Suppose

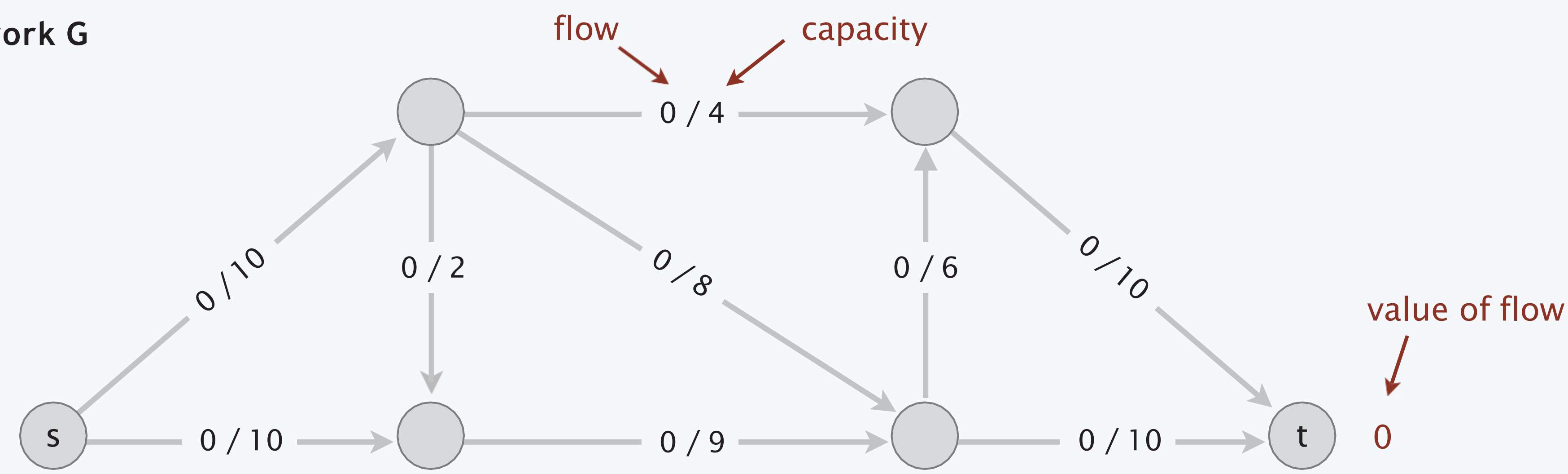
- f is an existing flow in G
- P is a simple s-t path
- *bottleneck(P, f)* is the minimum residual capacity of any edge on P, with respect to the flow f.

Consider the next procedure:

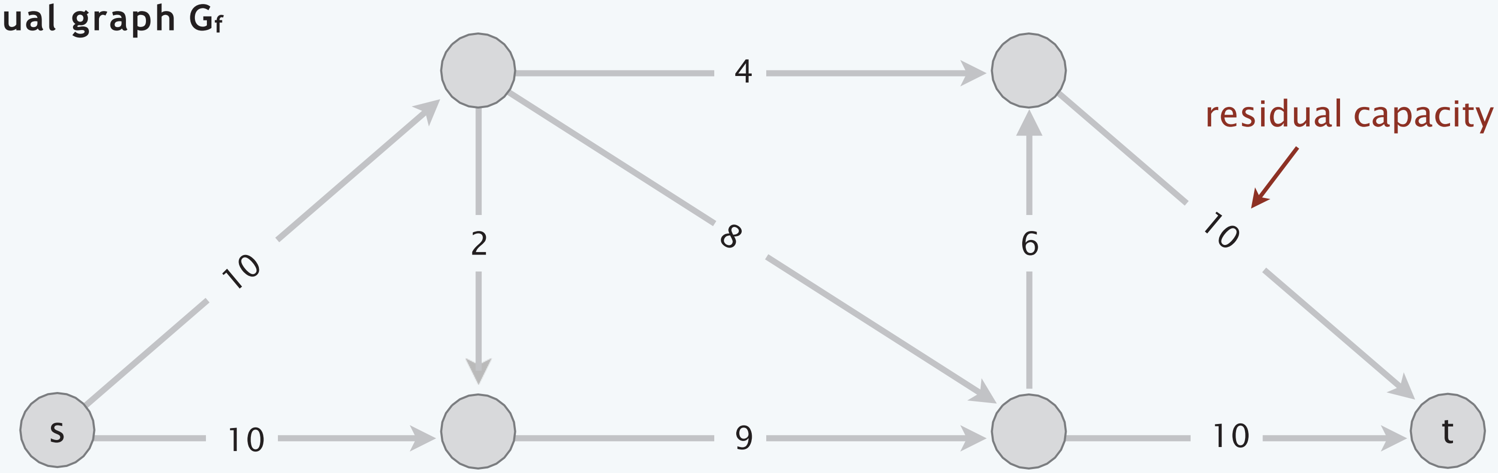
```
augment(f, P)
  Let b = bottleneck(P, f)
  For each edge (u, v) ∈ P
    If e = (u, v) is a forward edge then
      increase f(e) in G by b

    Else ((u, v) is a backward edge, and let e = (v, u))
      decrease f(e) in G by b
    Endif
  Endfor
  Return(f)
```

Augmenting paths demo

network G

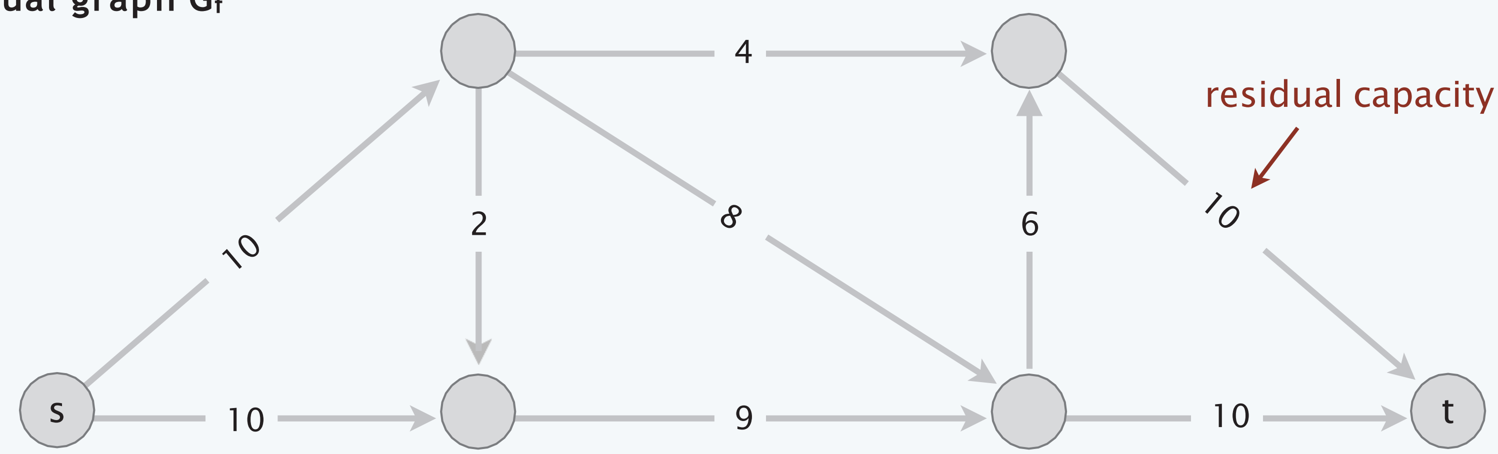


residual graph G_f

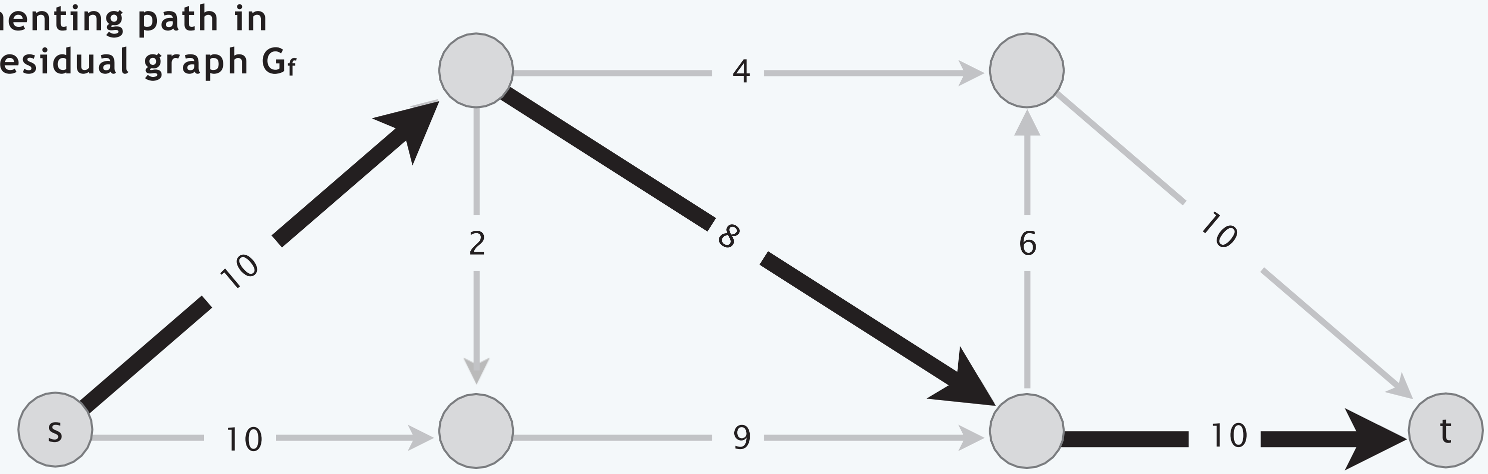


Augmenting paths demo

residual graph G_f

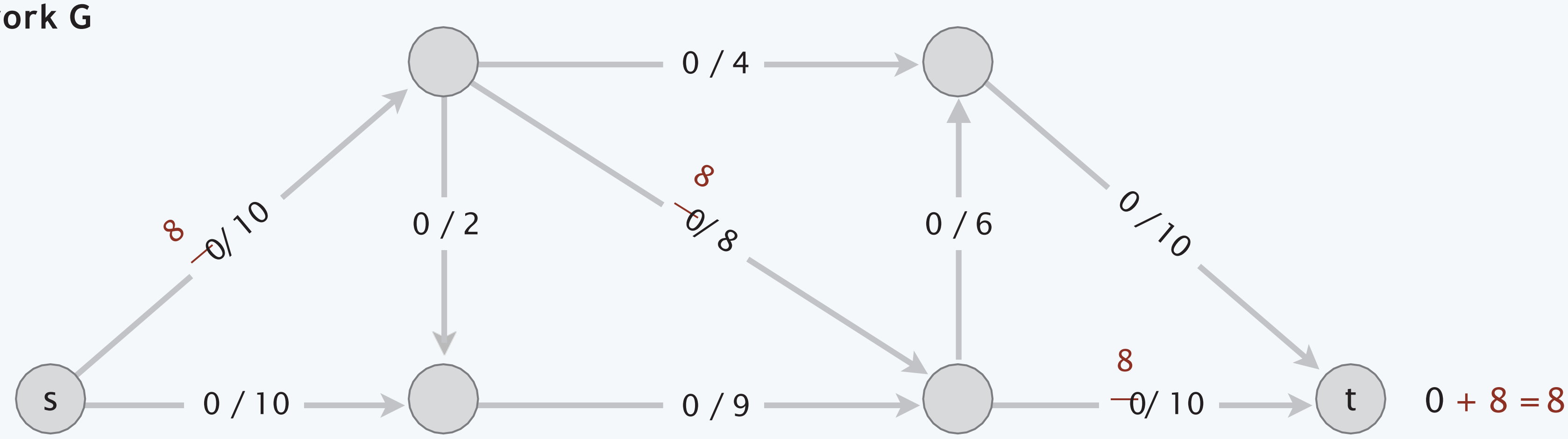


Augmenting path in the residual graph G_f

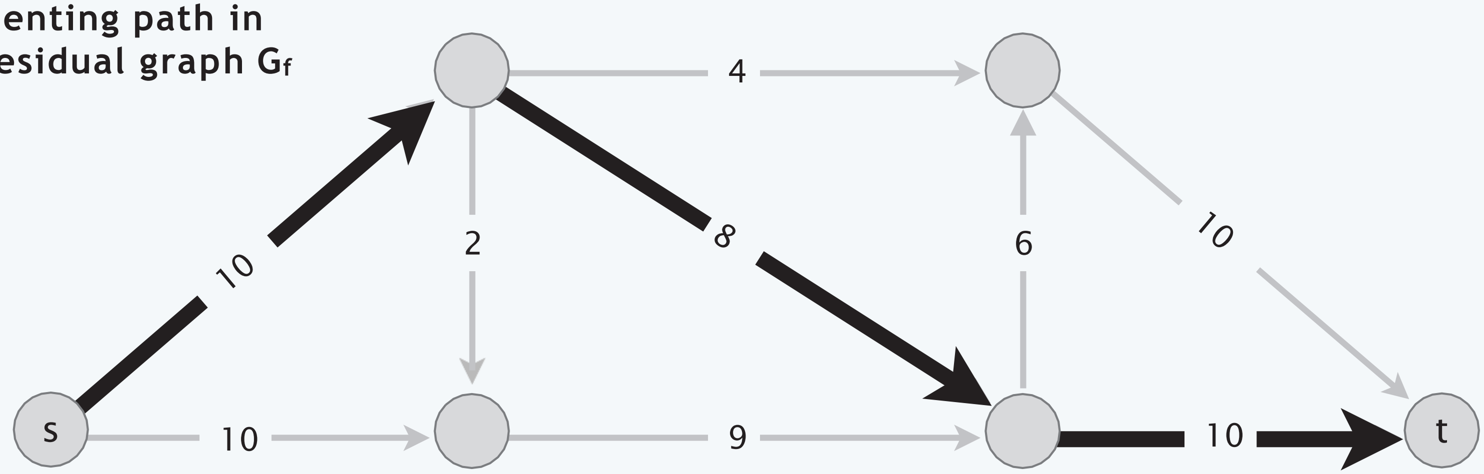


Augmenting paths demo

network G



Augmenting path in the residual graph G_f



Lemma: *Let f' be the flow resulting from Augment(f,P). Then f is a valid flow in G.*

proof?

Lemma: *Let f' be the flow resulting from Augment(f,P). Then f is a valid flow in G.*

proof: *Flow f' differs from f only on the edges of P, so only need to check these. Let e=(u,v) be an edge on P.*

- *If e is a forward edge, we avoided increasing above the capacity of the edge.*
- *If it's a backward edge, we avoided decreasing the flow on e below zero.*

More formally, if $e = (u,v)$ is a forward edge, then its residual capacity is $c_e - f(e)$. Thus:

$$0 \le f(e) \le f'(e) = f(e) + \texttt{bottleneck}(P, f) \le f(e) + (c_e - f(e)) = c_e$$

If $e = (u,v)$ is a backward edge arising from edge $e = (v,u)$, then its residual capacity is $f(e)$. Thus:

$$c_e \ge f(e) \ge f'(e) = f(e) - \texttt{bottleneck}(P, f) \ge f(e) - f(e) = 0$$

Ford-Fulkerson Algorithm

Ford-Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path P in the residual graph G_f.
- Augment flow along path P.
- Repeat until you get stuck.

Max-Flow
  Initially $f(e) = 0$ for all e in G
  While there is an s–t path in the residual graph G_f
    Let P be a simple s–t path in G_f
    $f' = \text{augment}(f, P)$
    Update f to be f'
    Update the residual graph G_f to be $G_{f'}$
  Endwhile
  Return f

Lemma 1 *At every intermediate stage of the Ford-Fulkerson Algorithm, the flow values $\{f(e)\}$ and the residual capacities in G_f are integers.*

Lemma 1 *At every intermediate stage of the Ford-Fulkerson Algorithm, the flow values $\{f(e)\}$ and the residual capacities in G_f are integers.*

Proof: *By induction on the number of iterations.*

Basis step: The statement is clearly true before any iterations of the While loop.

IH: Suppose it is true after j iterations.

IS: Since all residual capacities in G_f are integers, the value bottleneck(P, f) for the augmenting path found in iteration $j + 1$ will be an integer.

Thus the flow f' will have integer values => so will the capacities of the new residual graph.

Lemma 2:

Let f be a flow in G, and let P be a simple s-t path in G_f. Then $\nu(f') = \nu(f) + \texttt{bottleneck}(P, f)$; and since $\texttt{bottleneck}(P, f) > 0$, we have $\nu(f') > \nu(f)$.

Lemma 2:

Let f be a flow in G, and let P be a simple s-t path in G_f. Then $\nu(f') = \nu(f) + \texttt{bottleneck}(P, f)$; and since $\texttt{bottleneck}(P, f) > 0$, we have $\nu(f') > \nu(f)$.

Proof: *The first edge e of P must be an edge out of s in the residual graph Gf .*

The path is simple => it does not visit s again.

Since G has no edges entering s, the edge e must be a forward edge.

We increase the flow on this edge by bottleneck(P, f), and do not change the flow on any other edge incident to s. Therefore the value of f ' exceeds the value of f by bottleneck(P, f).

Let $C = \sum_{e \text{ out of } s} c_e$

Lemma 3: *Suppose that all capacities in the flow network G are integers. Then the Ford-Fulkerson Algorithm terminates in at most C iterations of the While loop.*

Let $C = \sum_{e \text{ out of } s} c_e$

Lemma 3: *Suppose that all capacities in the flow network G are integers. Then the Ford-Fulkerson Algorithm terminates in at most C iterations of the While loop.*

Proof: *No flow in G can have value greater than C, due to the capacity condition on the edges leaving s.*

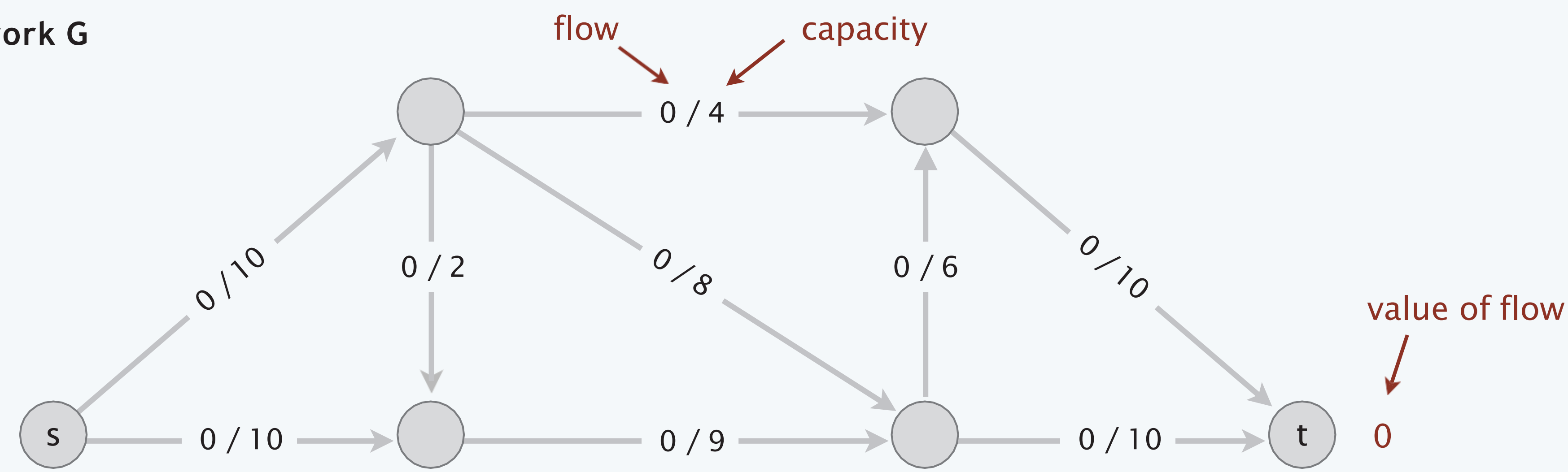
By Lemma 2, the value of the flow maintained by the Ford-Fulkerson Algorithm increases in each iteration.

Then by Lemma 1, it increases by at least 1 in each iteration.

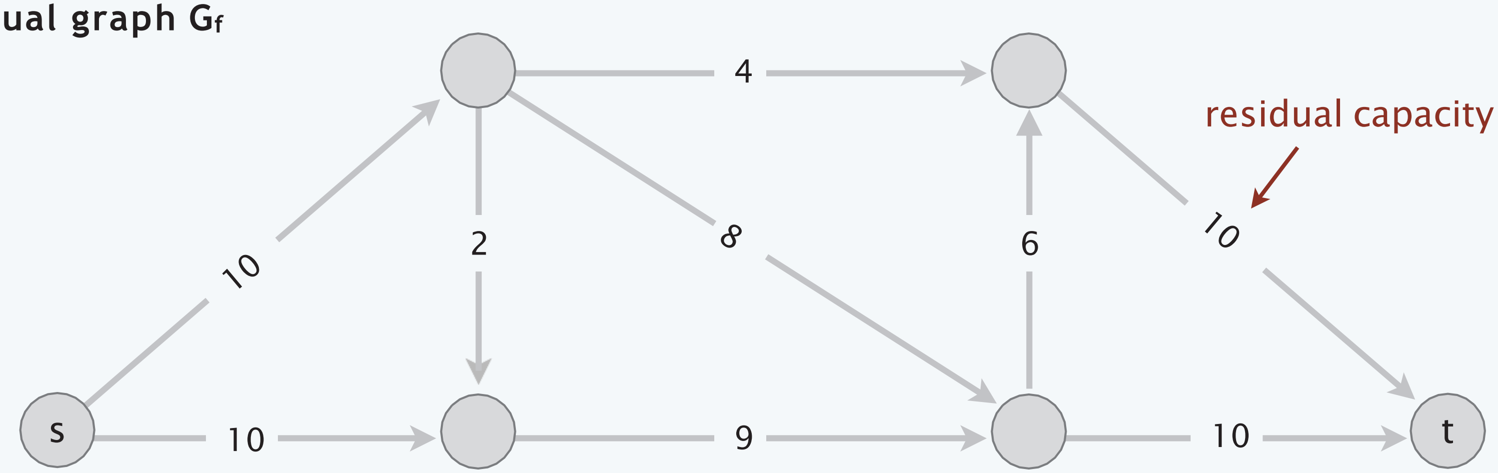
Since it starts with the value 0, and cannot go higher than C, the While loop in the Ford-Fulkerson Algorithm can run for at most C iterations.

Ford-Fulkerson algorithm demo

network G

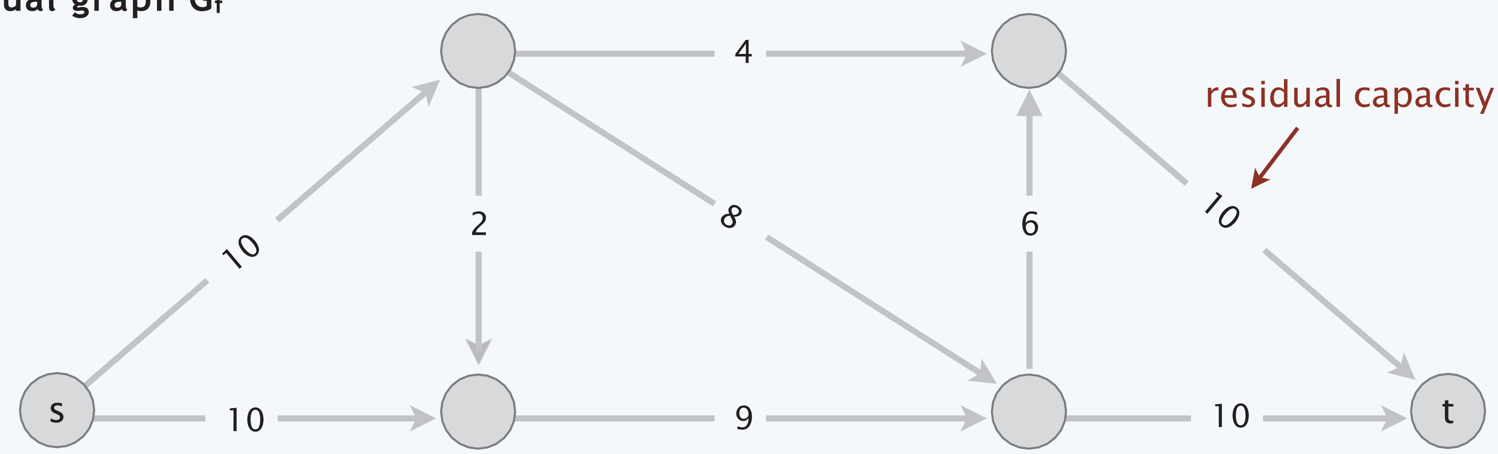


residual graph G_f

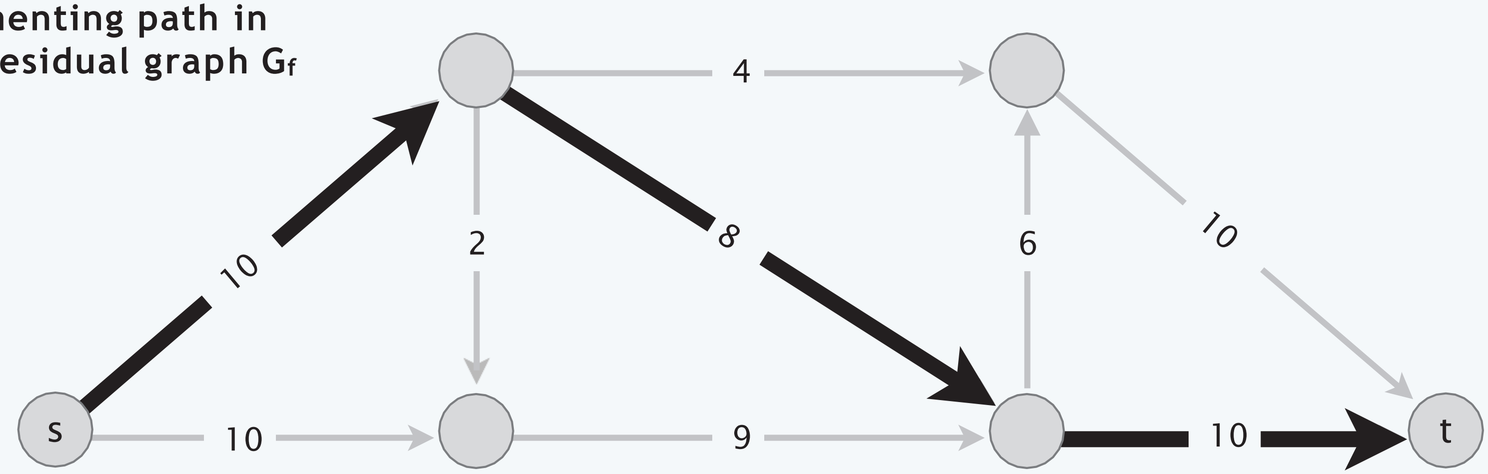


Ford-Fulkerson algorithm demo

residual graph G_f

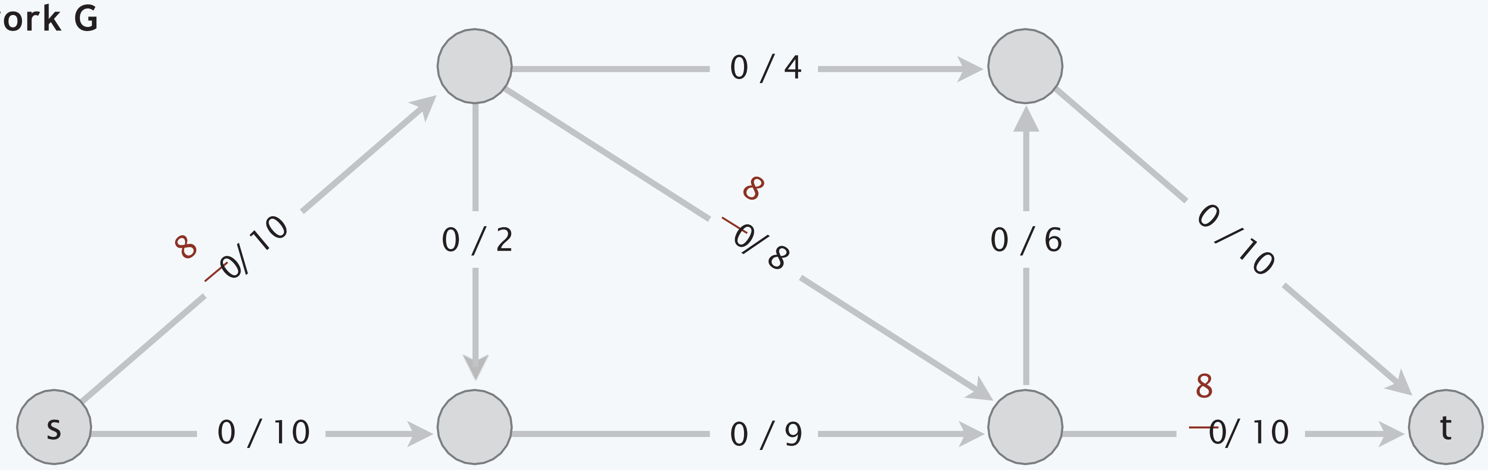


Augmenting path in the residual graph G_f



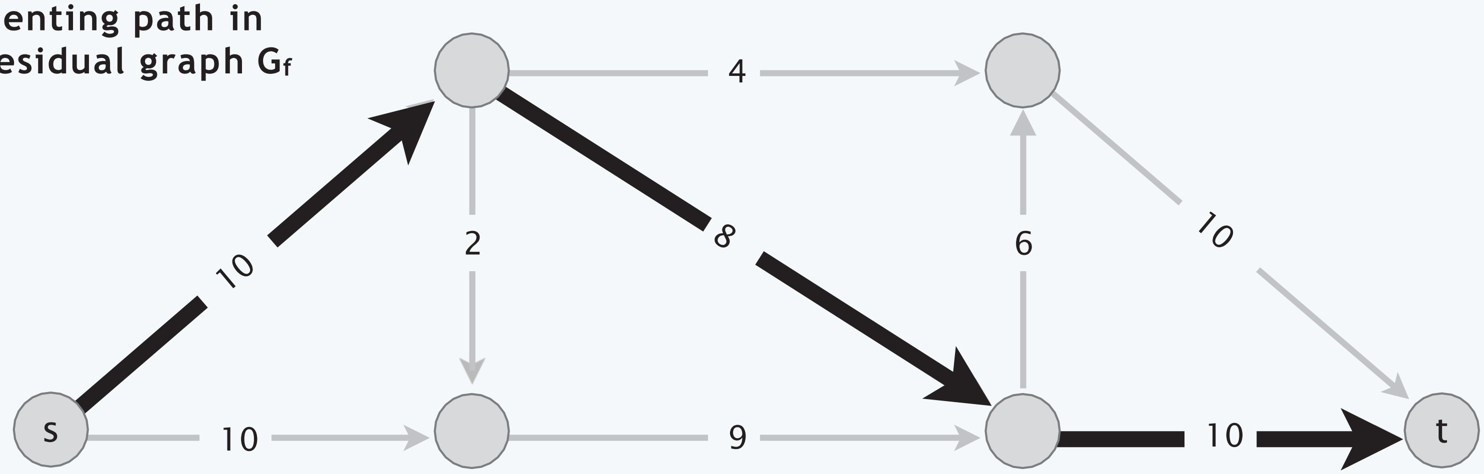
Ford-Fulkerson algorithm demo

network G



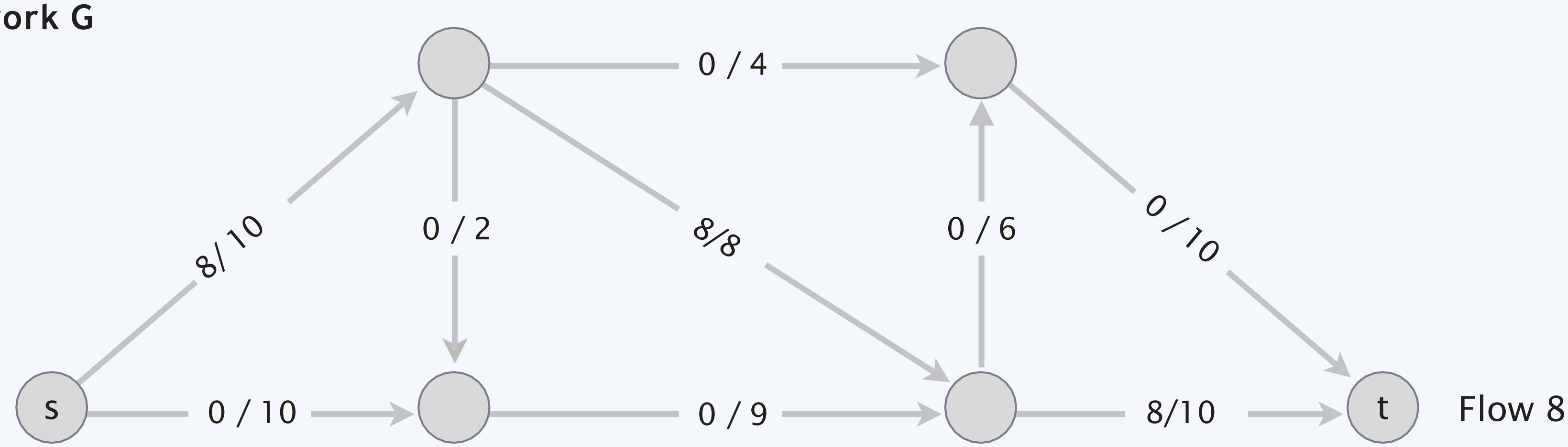
0 + 8 = 8

Augmenting path in the residual graph G_f

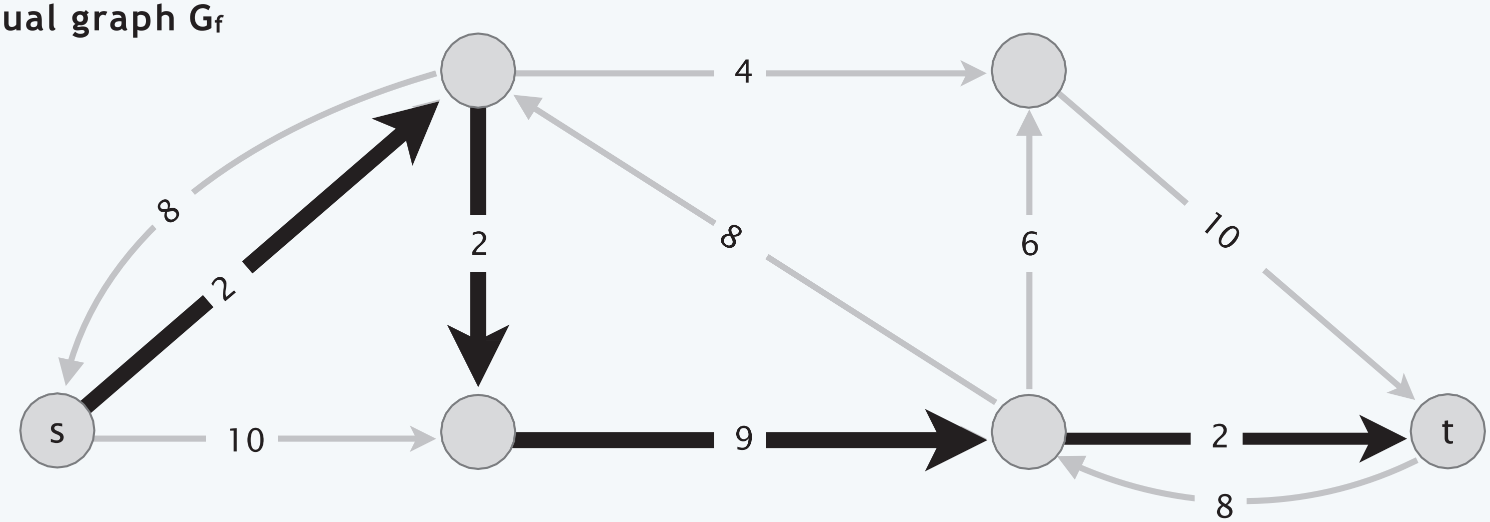


Ford-Fulkerson algorithm demo

network G

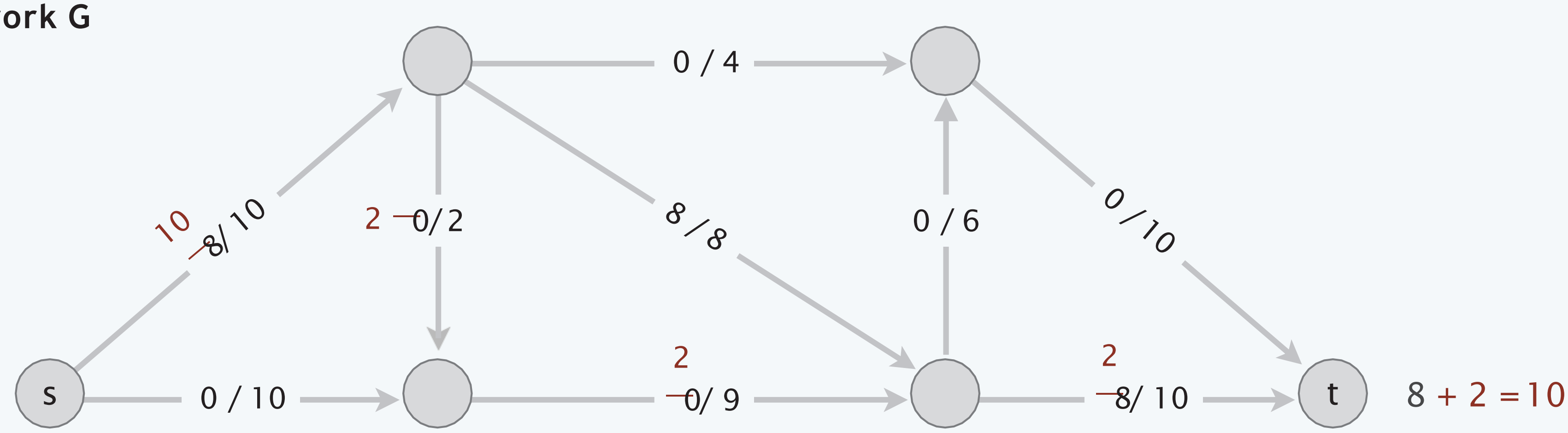


residual graph G_f

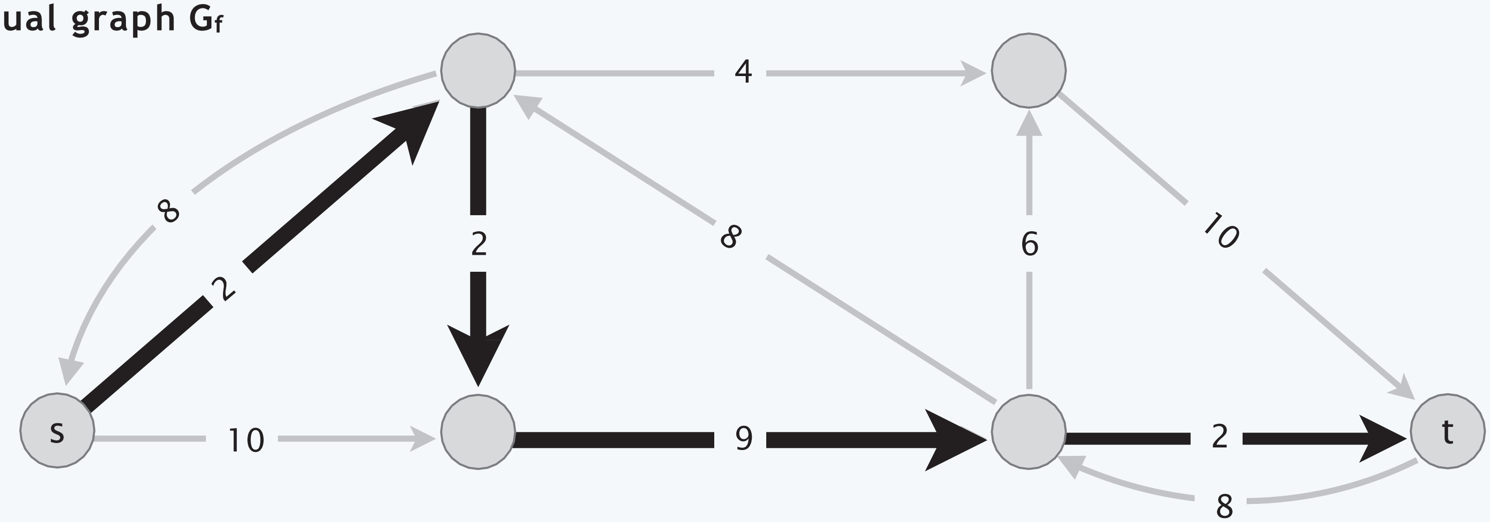


Ford-Fulkerson algorithm demo

network G

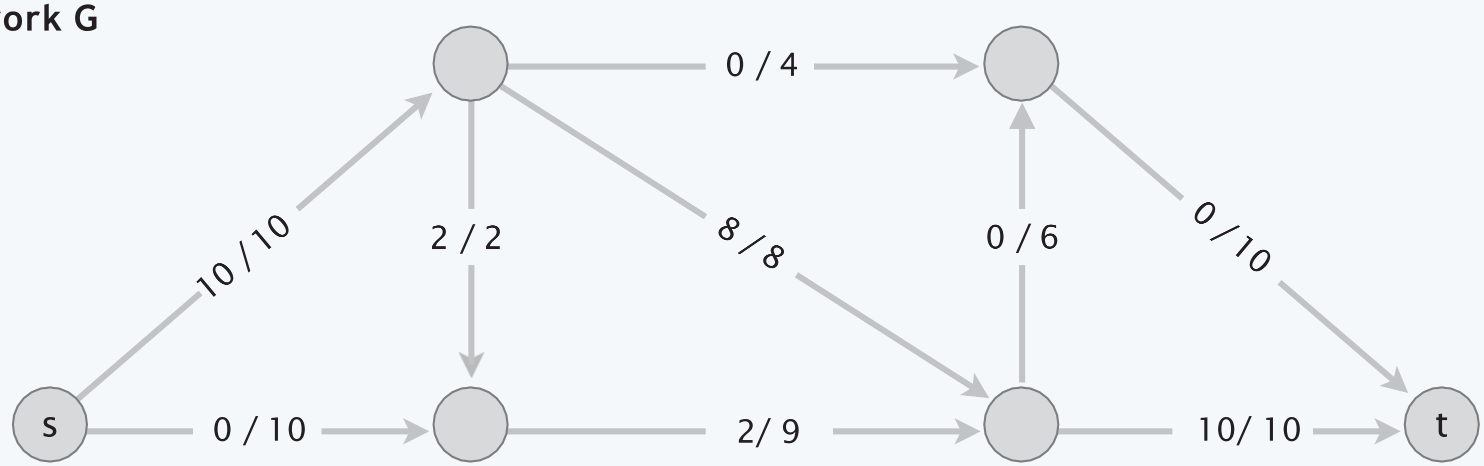


residual graph G_f



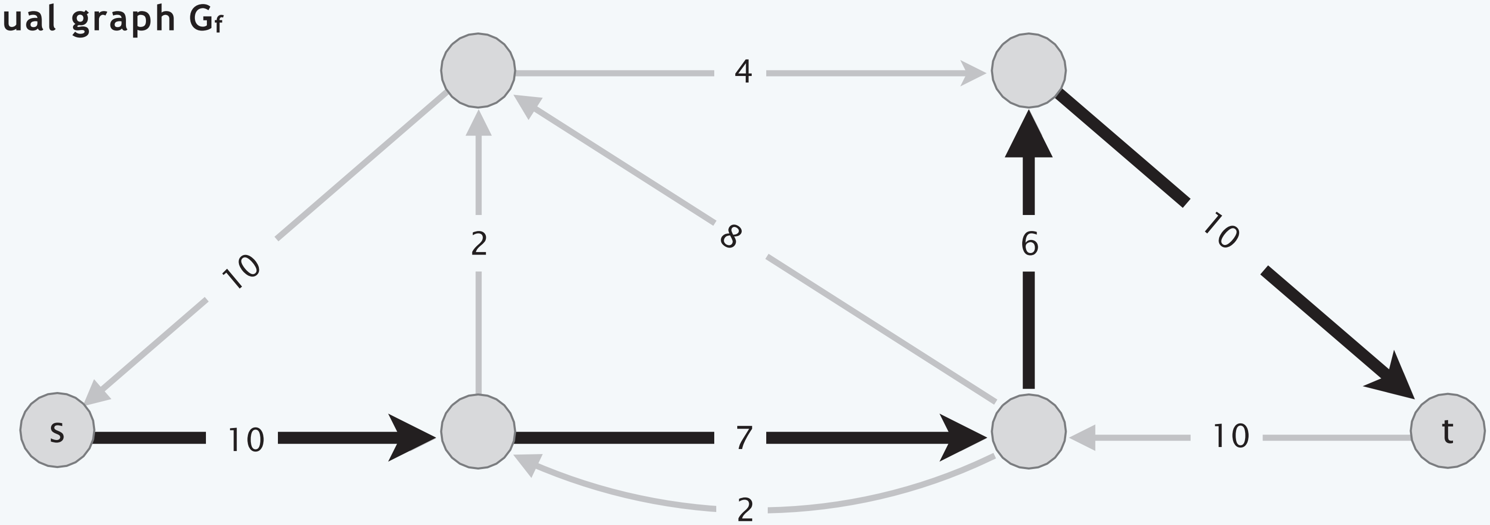
Ford-Fulkerson algorithm demo

network G



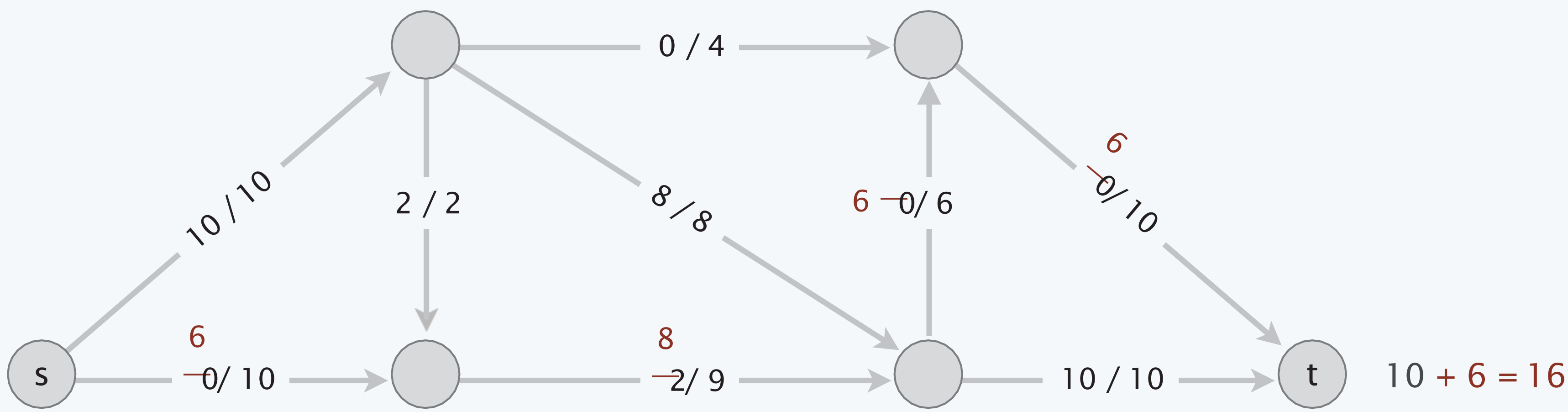
Flow 10

residual graph G_f

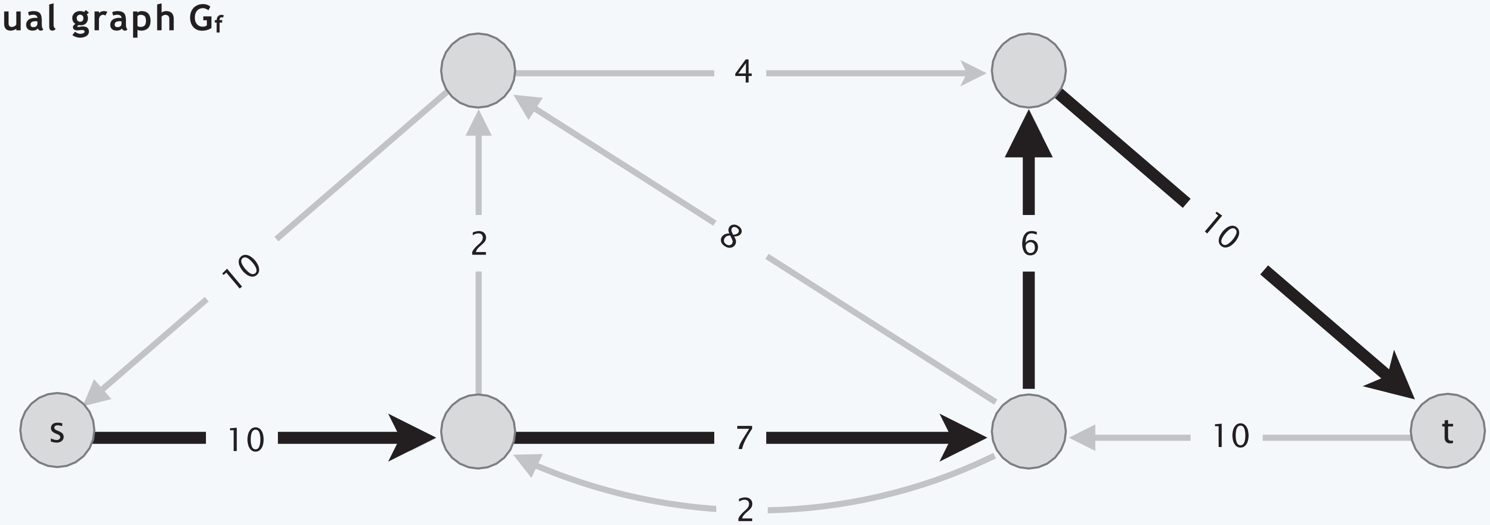


Ford-Fulkerson algorithm demo

network G

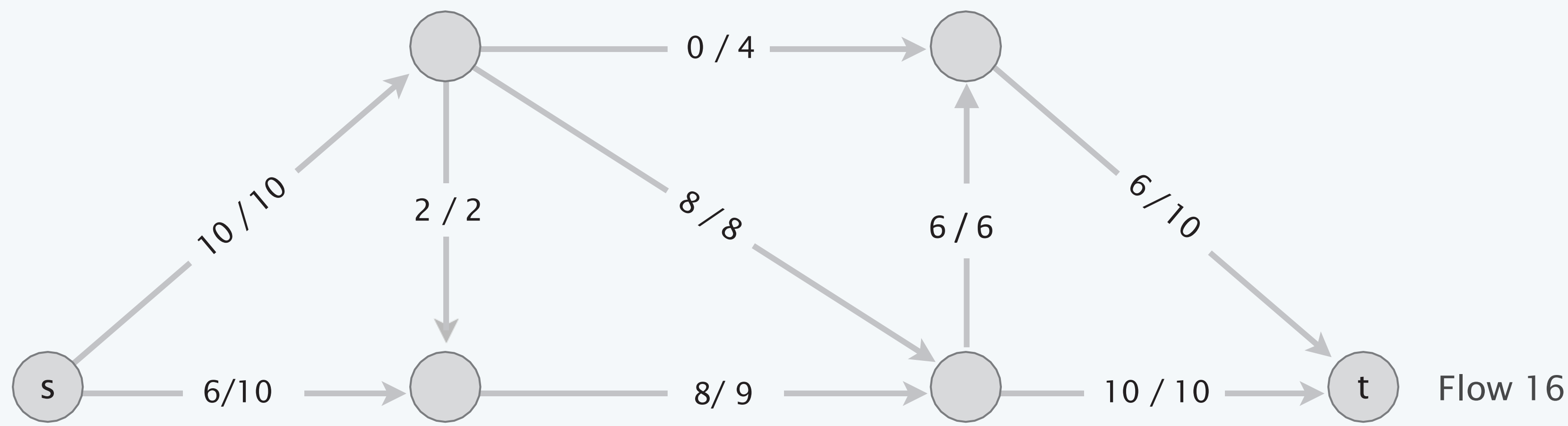


residual graph G_f

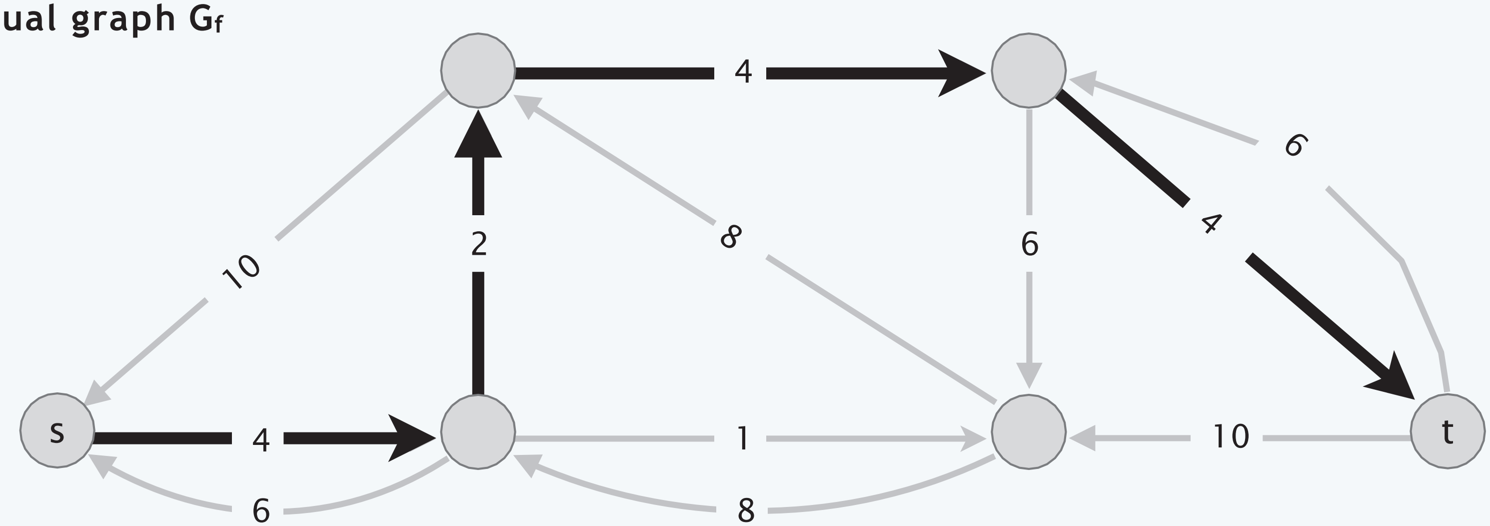


Ford-Fulkerson algorithm demo

network G

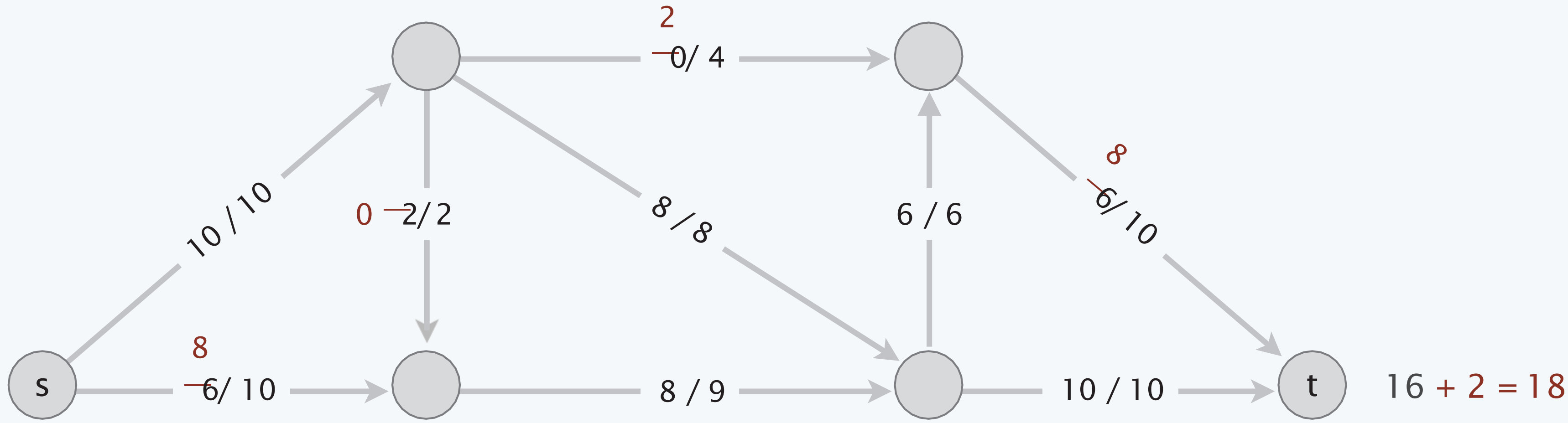


residual graph G_f

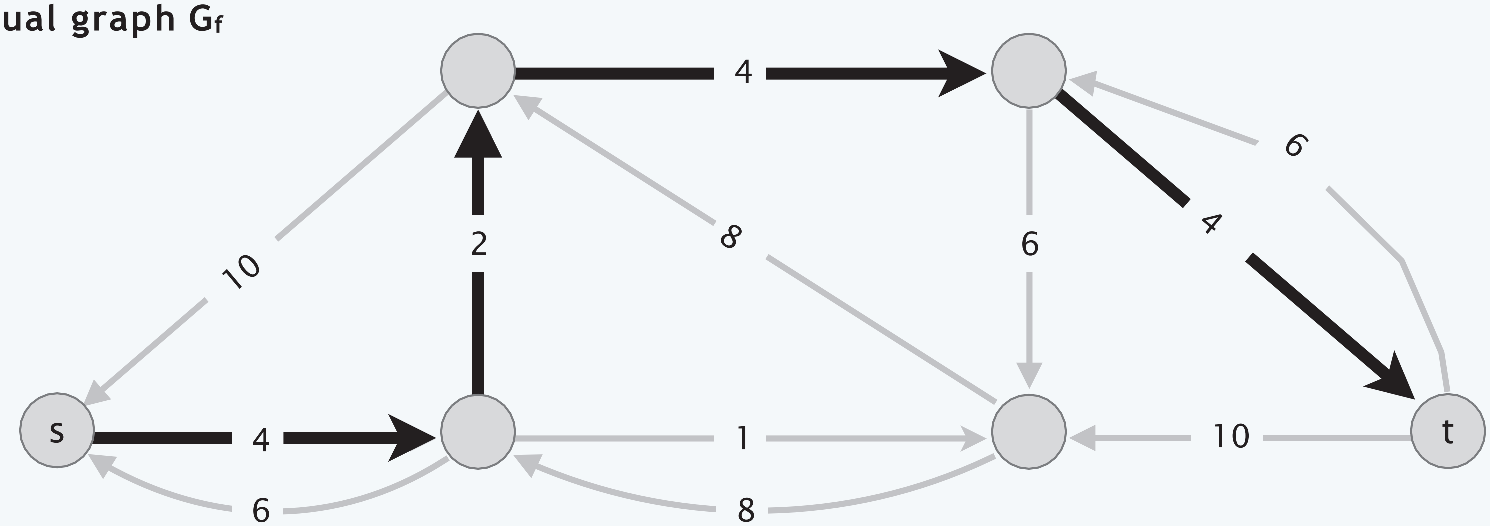


Ford-Fulkerson algorithm demo

network G

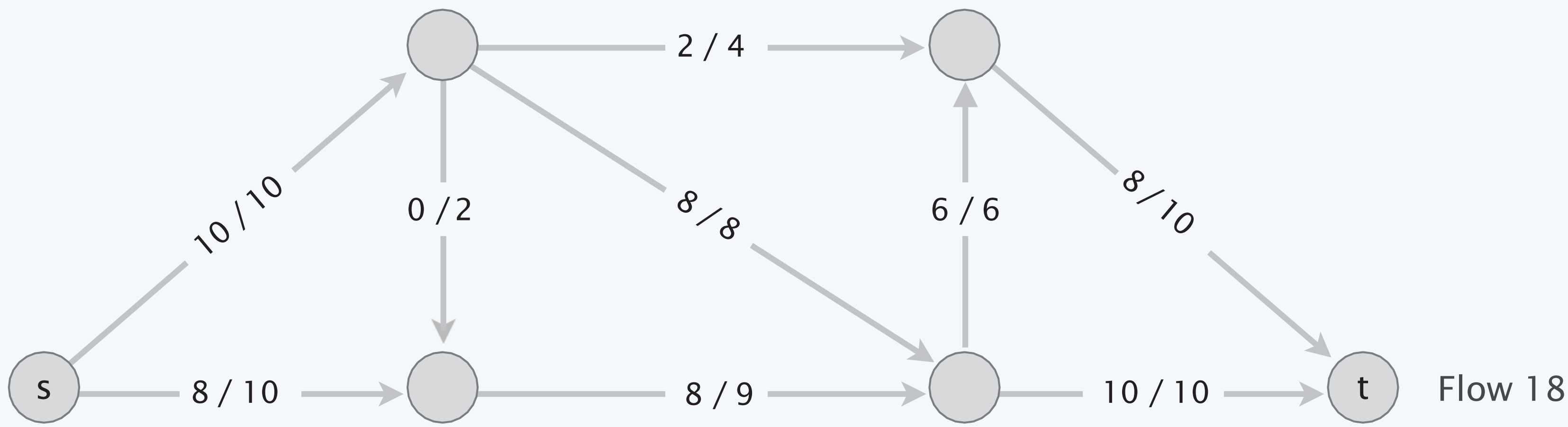


residual graph G_f

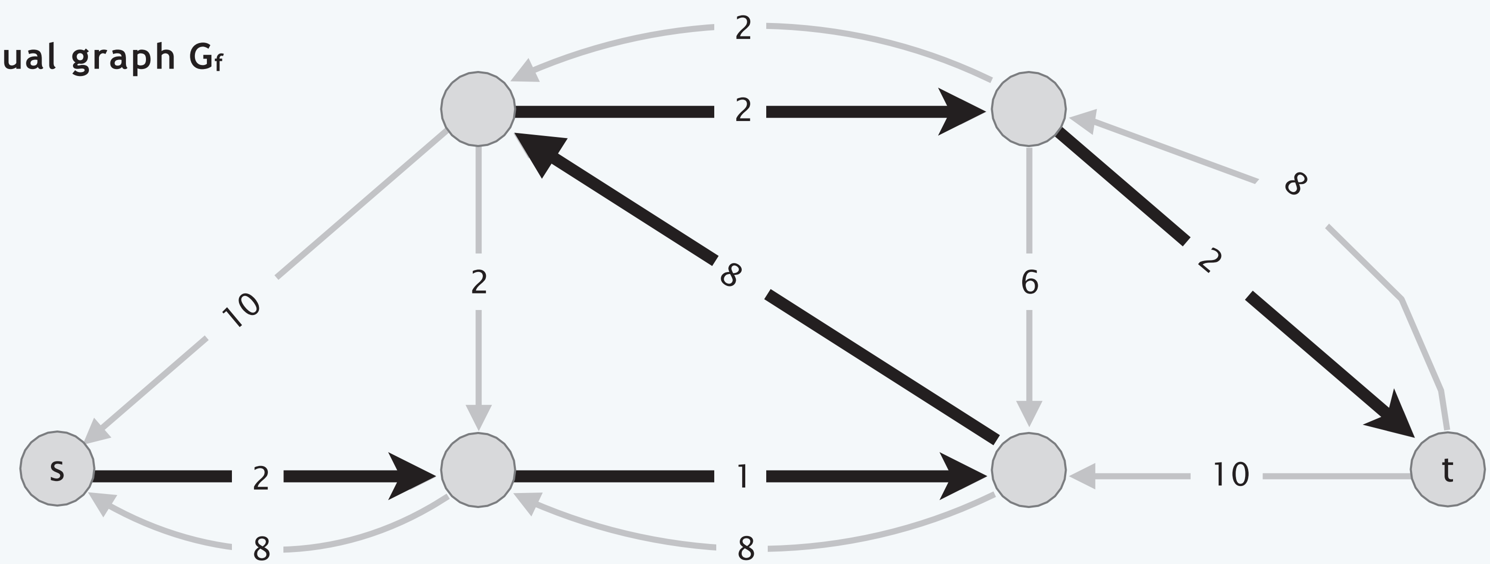


Ford-Fulkerson algorithm demo

network G

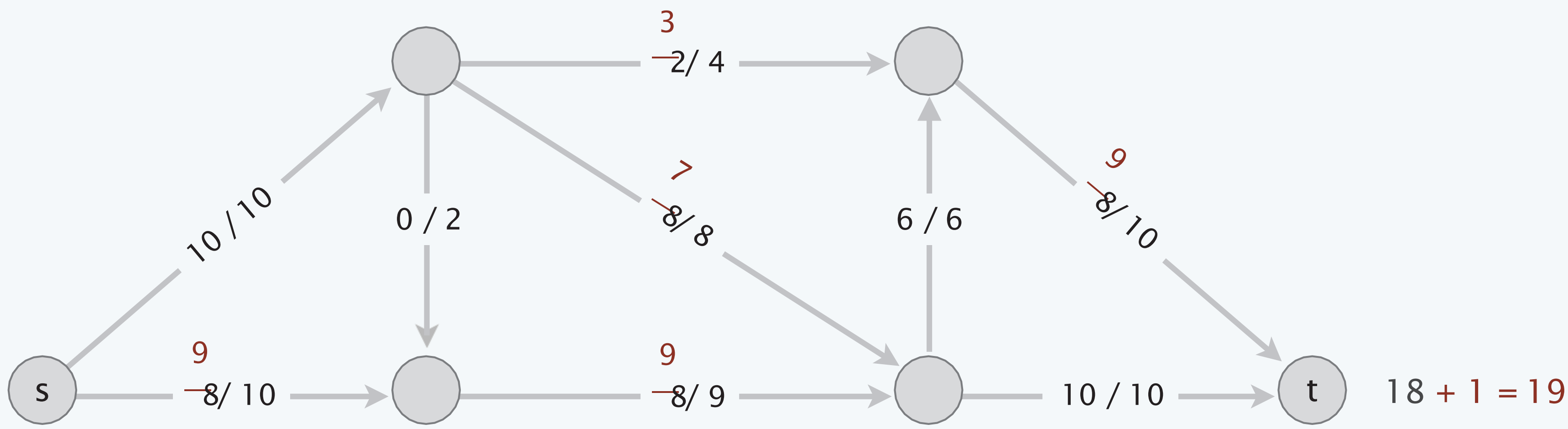


residual graph G_f



Ford-Fulkerson algorithm demo

network G



- *Run Ford-Fulkerson. Look at the final residual graph.*
- *Find the vertices reachable from the source in the residual graph.*
- *All edges which are from a reachable vertex to non-reachable vertex are minimum cut edges.*