Announcements

- HW 2 due yesterday, 6 pm
  - Can use at most two late days

- HW3 is out, and due in slightly more than two weeks

CS6161: Design and Analysis of Algorithms (Fall 2020)

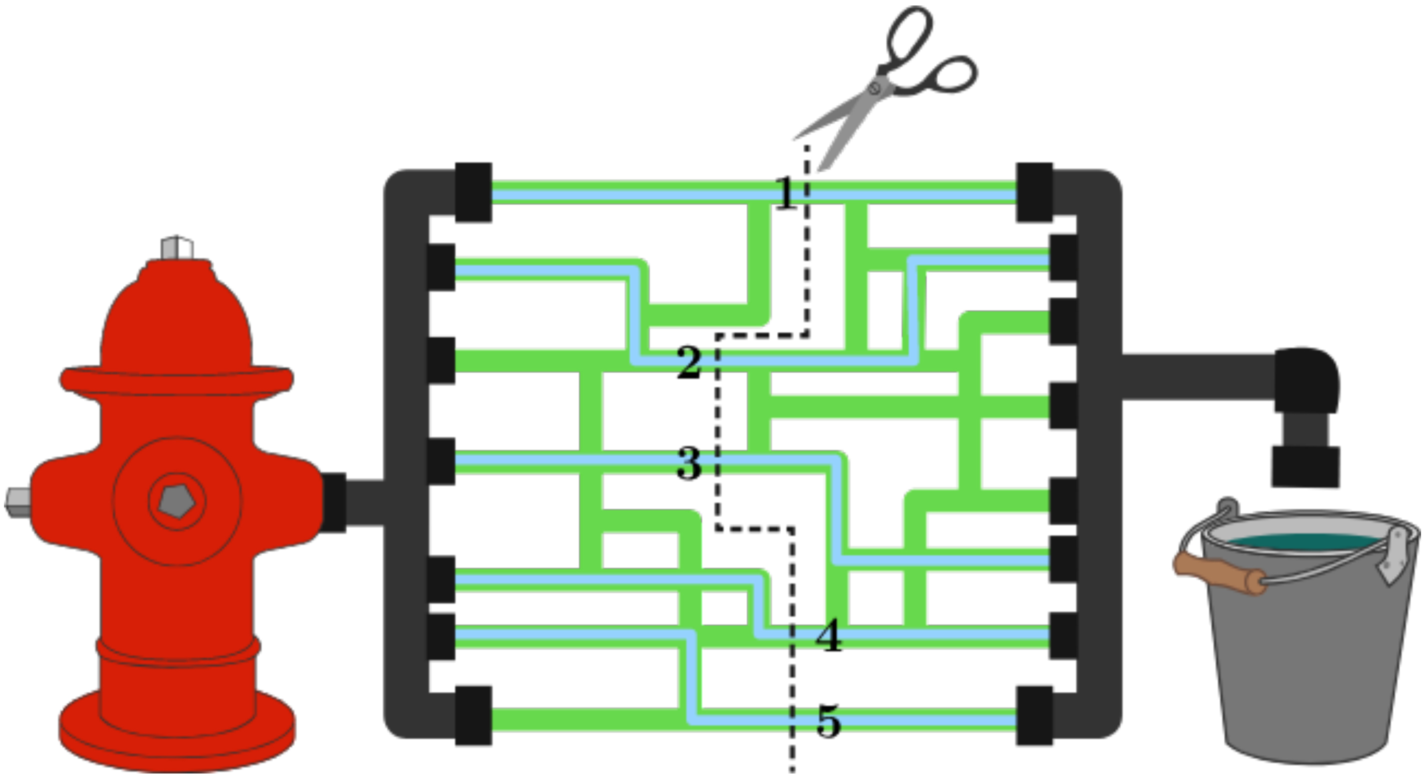
Max Flow and Min Cut (I)

Instructor: Haifeng Xu

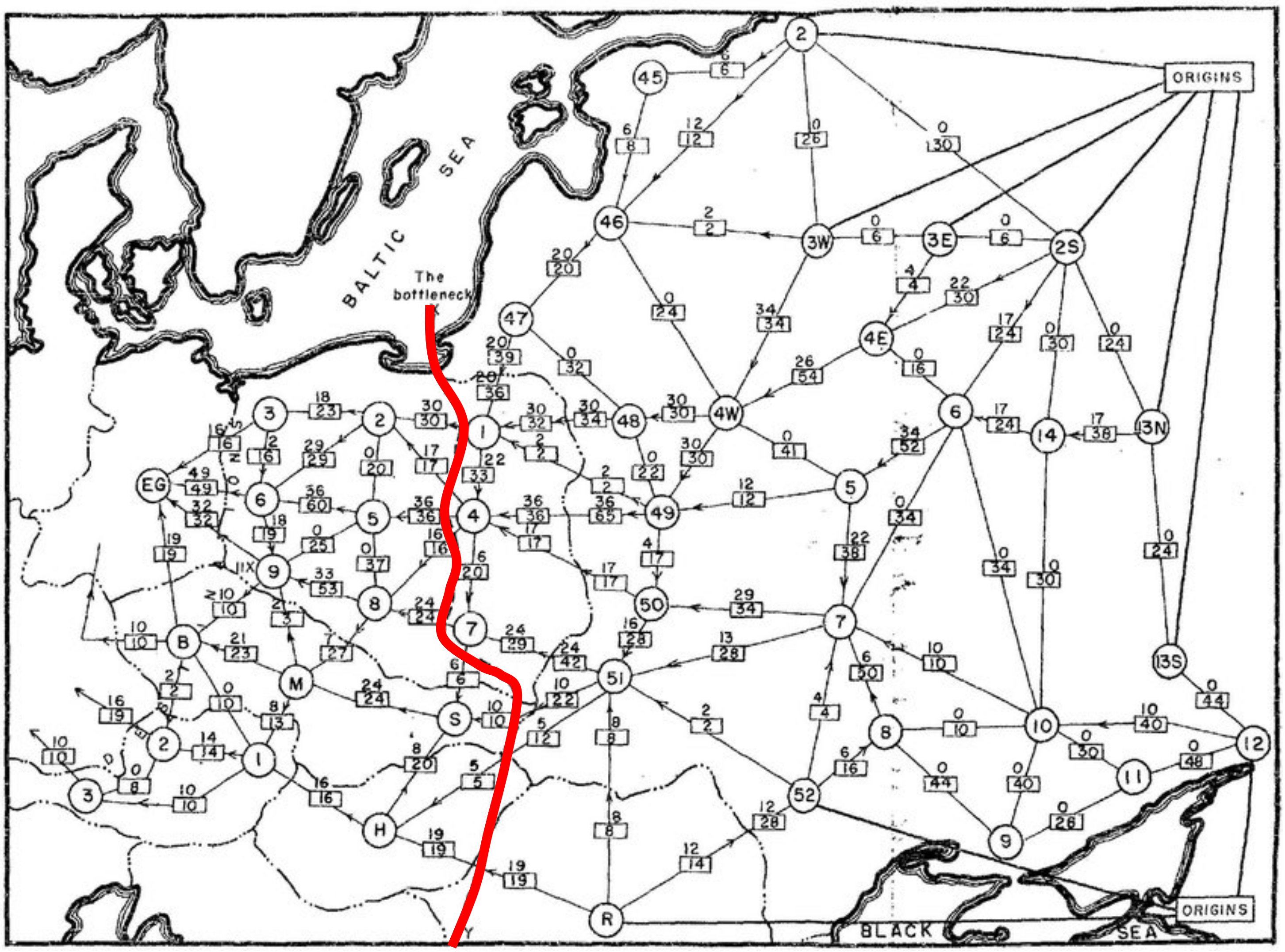
Outline

- Max Flow Problem and Min Cut Problem
- Ford-Fulkerson (FF) Algorithm

Max Flow and Min Cut Problems: Example 1



Max Flow and Min Cut Problems: Example 2



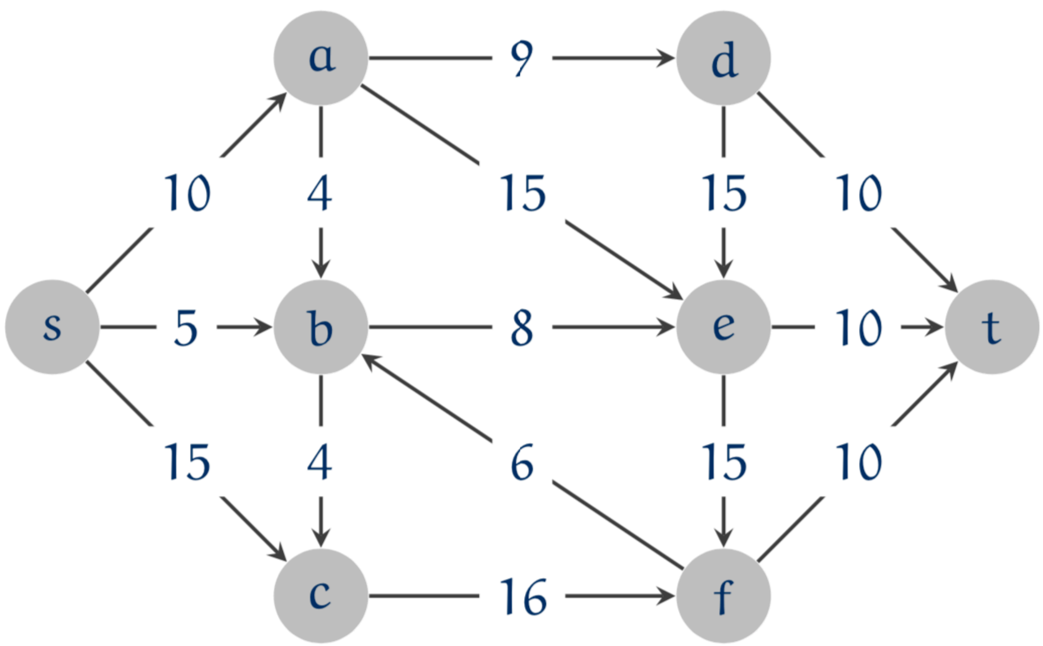
Turns out they are essentially the same problem

- Our first example of duality
- Duality theory, started by Von Neumann, is one of the most important theories in optimization, algorithms, economics, etc.

Flow Network

➢ A flow network is a directed graph $G = (V, E)$ with:

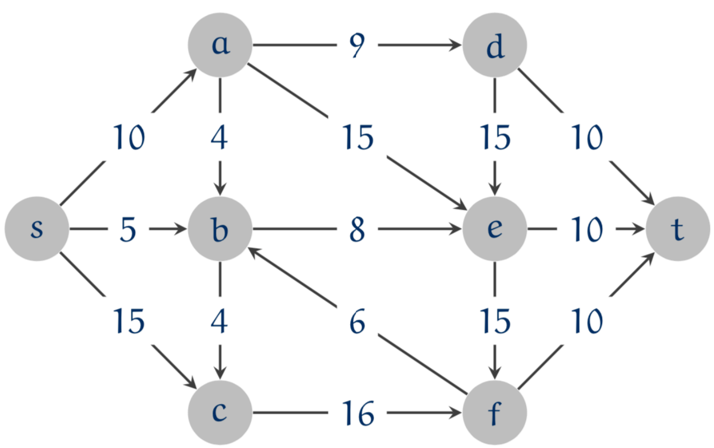
- A source $s \in V$
- A sink $t \in V$
- Non-negative capacity $c(e)$ for each edge $e \in E$



Note, $c(e)$ differs from cost in previous lectures

Flow Network

- A flow network is a directed graph $G = (V, E)$ with:
  - A source $s \in V$
  - A sink $t \in V$
  - Non-negative capacity $c(e)$ for each edge $e \in E$
- Remarks
  - Abstraction for material flowing from s to t through edges
  - Always assume directed edges
  - Assume no parallel edges <u>for simplicity</u>
  - Assume no edges entering s since these edges will never be used (as we will see later)
  - Similarly, no edges exiting t



Flow in Flow Networks

- An s-t flow $f: E \to \mathbb{R}$ is a real-valued function that satisfies:
  - $\forall\, e \in E : 0 \le f(e) \le c(e)$ [capacity]
  - $\forall\, v \neq \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]
- Th value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e)$

Remarks

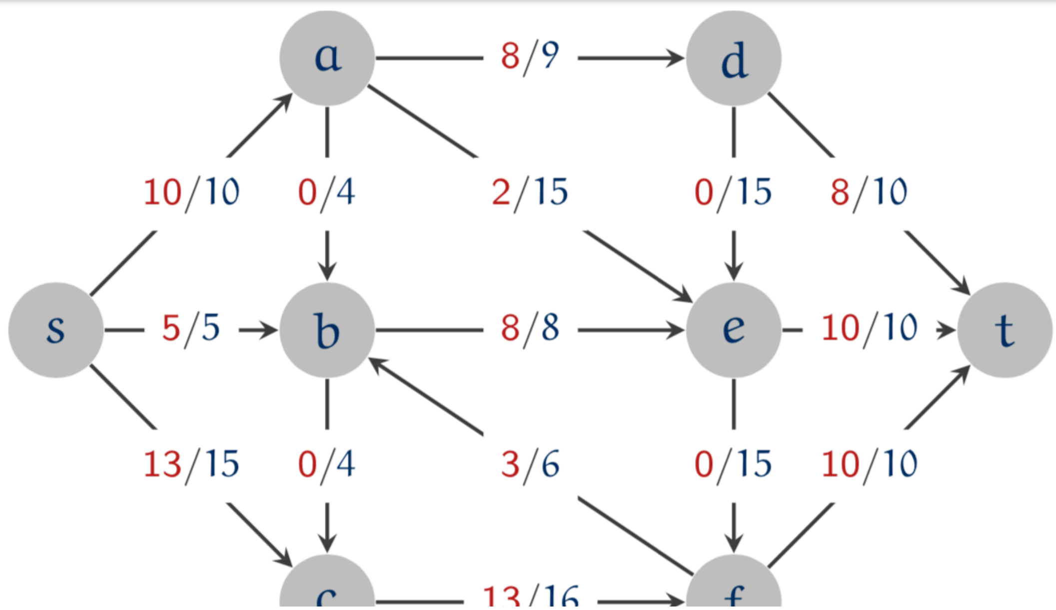
- $val(f)$ also equals $\sum_{e \text{ into } t} f(e)$
- This is the amount of flow going from s to t

Flow in Flow Networks

- An *s-t flow* $f: E \to \mathbb{R}$ is a real-valued function that satisfies:
  - $\forall\, e \in E : 0 \le f(e) \le c(e)$ [capacity]
  - $\forall\, v \ne \{s,t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]
- Th value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e)$

Max-Flow Problem

Find a (feasible) flow of maximum value

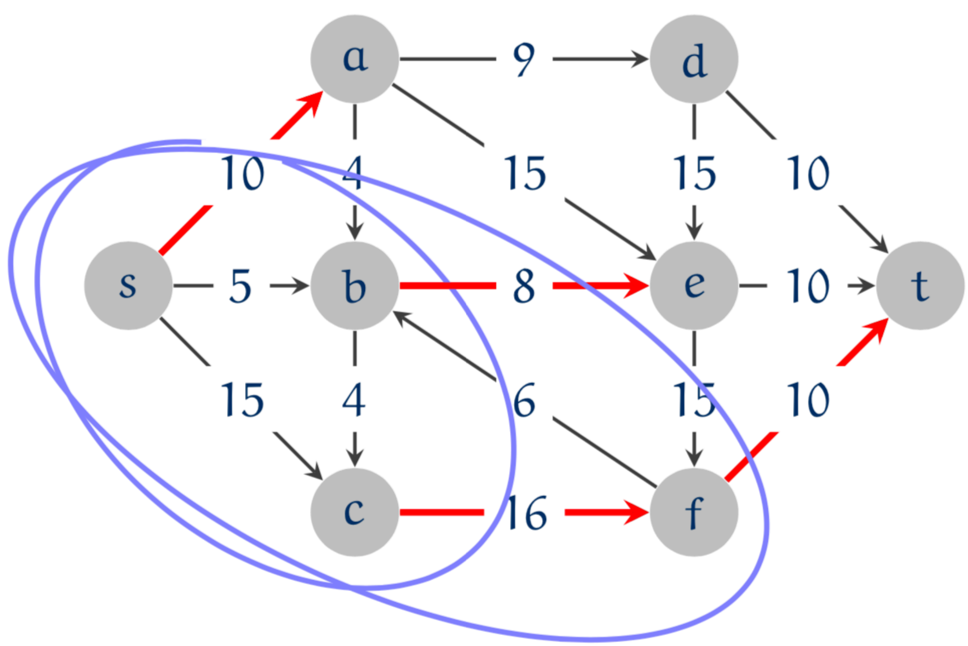


Cut in Flow Networks

- An s-t cut is a partition (A, B) of the vertices with $s \in A$ and $t \in B$
- The capacity of a cut = sum of capacities of edges from A to B.

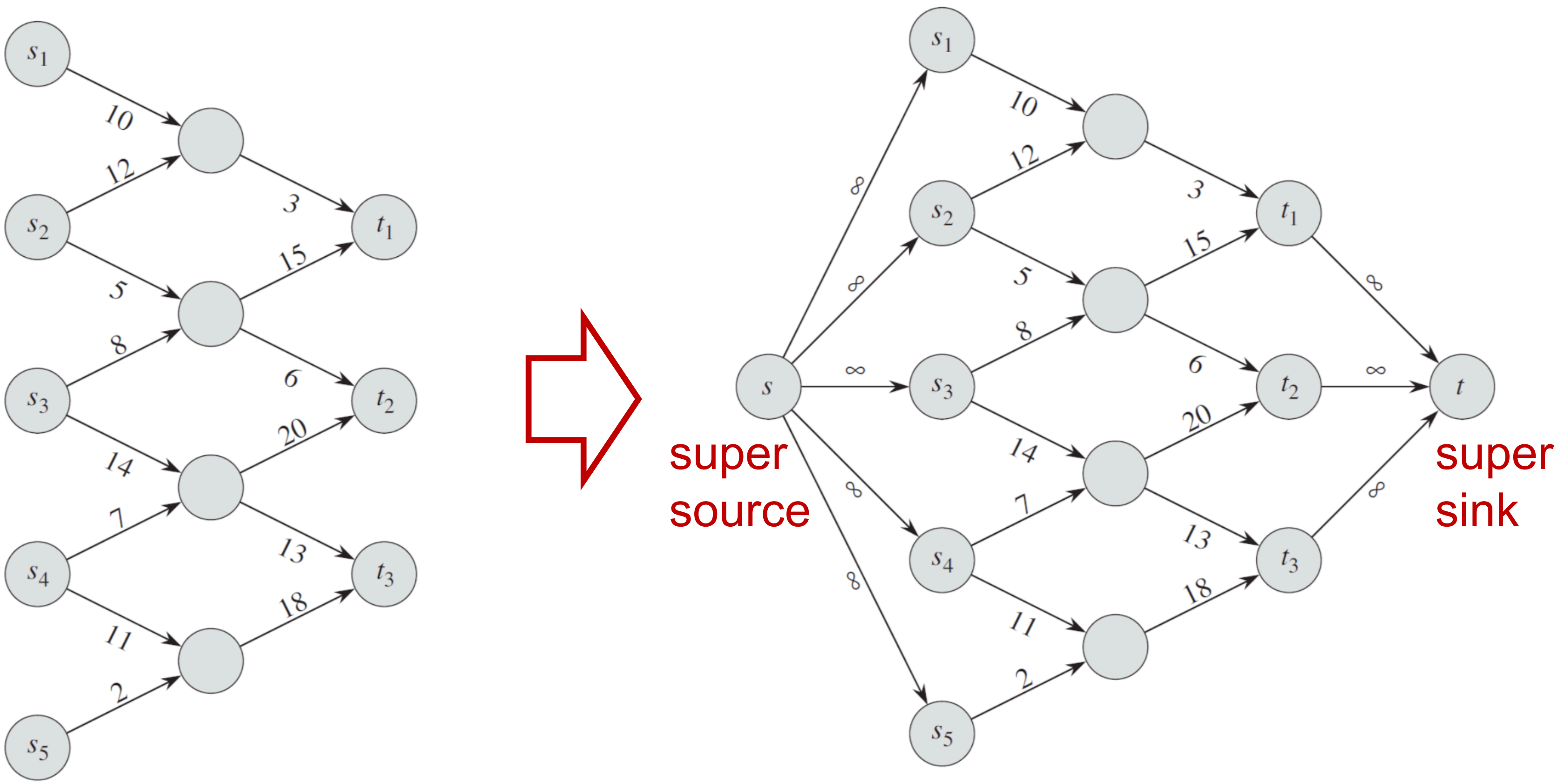
Min-Cut Problem

Find a cut of minimum value



What About Multiple Sources and Sinks?

Want to flow maximum amount of flows from sources to sinks



Can be reduced to single-source single-sink case

Applications

Really a lot – Max Flow problem models the situation where goods need to be transported through a network with limited capacities

- bipartite matching
- disjoint paths
- airline scheduling
- image segmentation
- project selection
- baseball elimination
- etc.

Will discuss some of them later

Outline

- Max Flow Problem and Min Cut Problem
- Ford-Fulkerson (FF) Algorithm

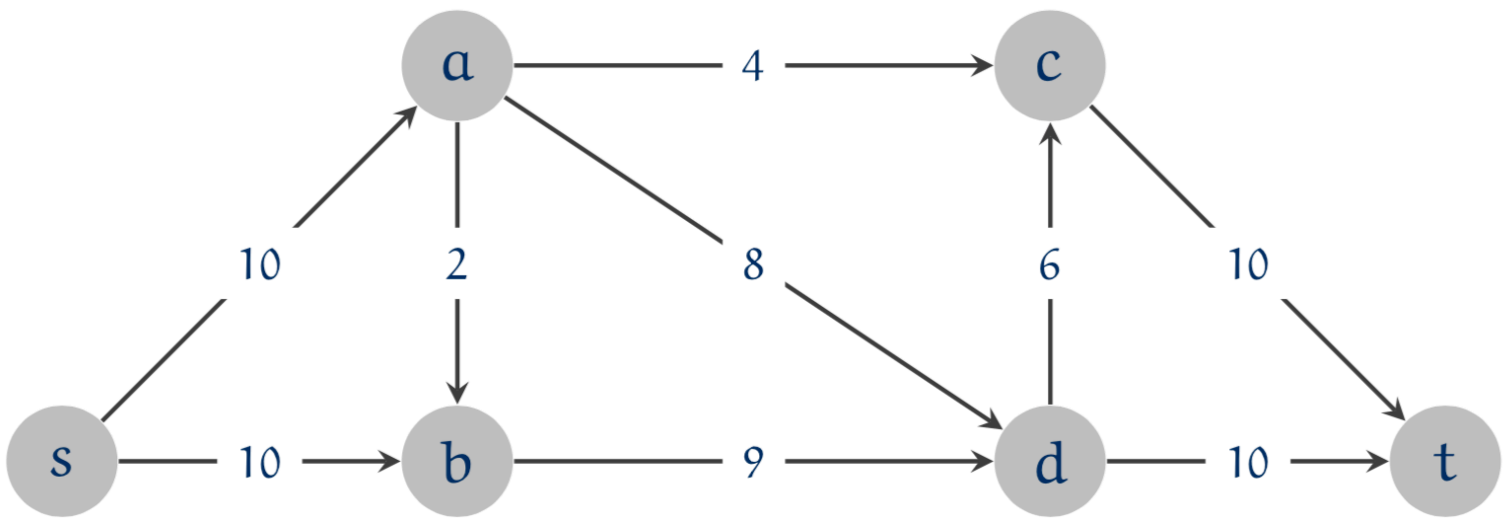
Towards a Max-Flow Algorithm

What would be your first try?

Greedy algorithm

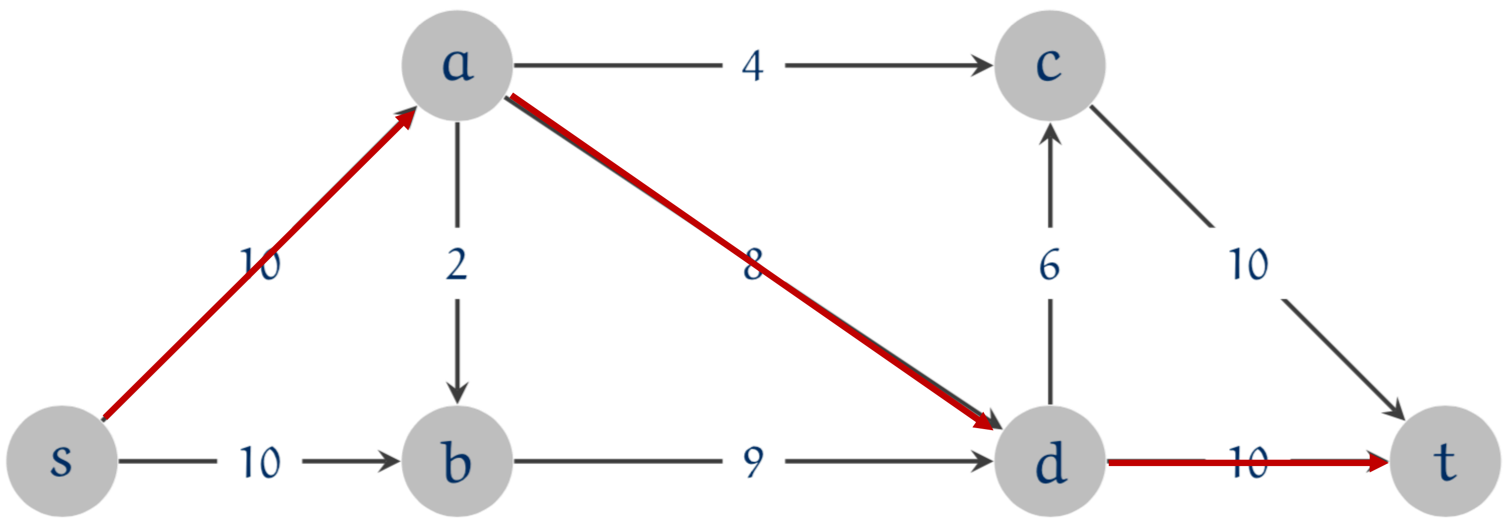
- Start with $f(e) = 0$ for all edges $e \in E$
- Find an $s - t$ path P using edges with $f(e) < c(e)$
- Augment flow along path P until some edge's capacity is reached
- Repeat, until stuck

Demo: Simple Greedy Fails



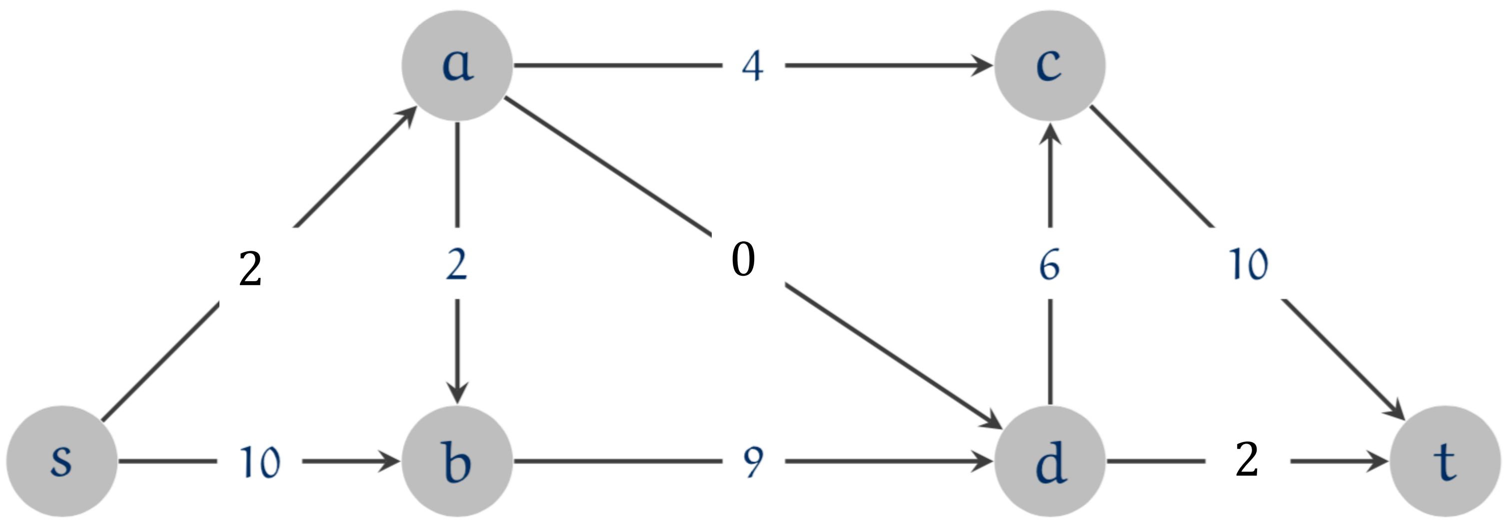
$Val(f) = 0$

Demo: Simple Greedy Fails



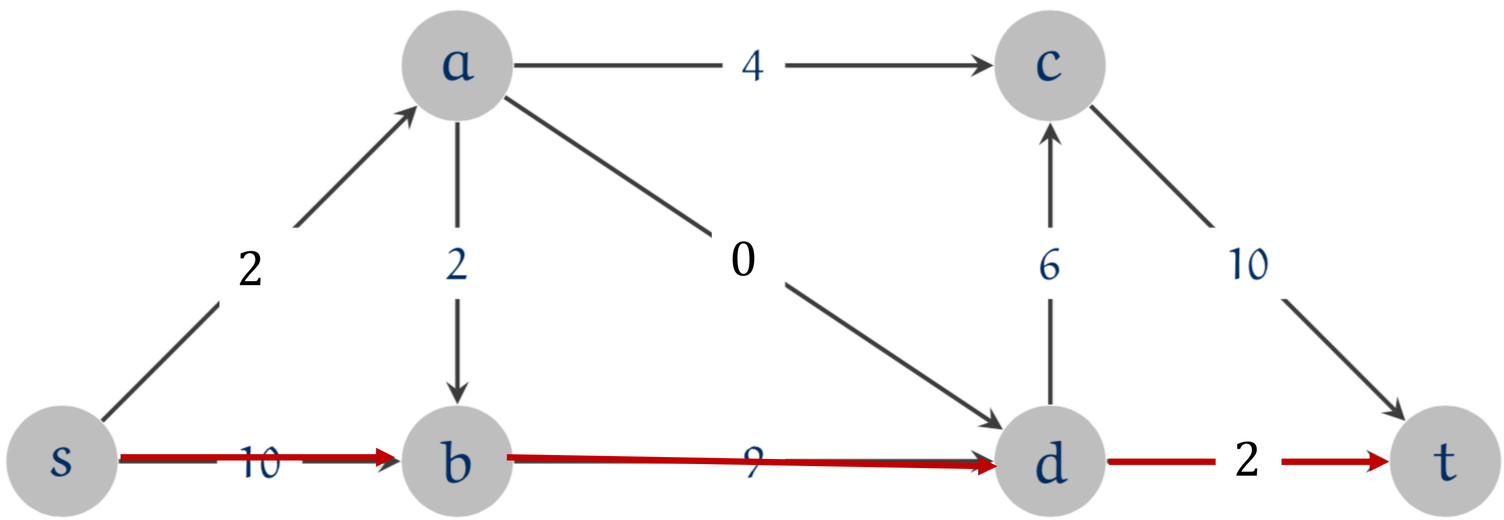
$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8$

Demo: Simple Greedy Fails



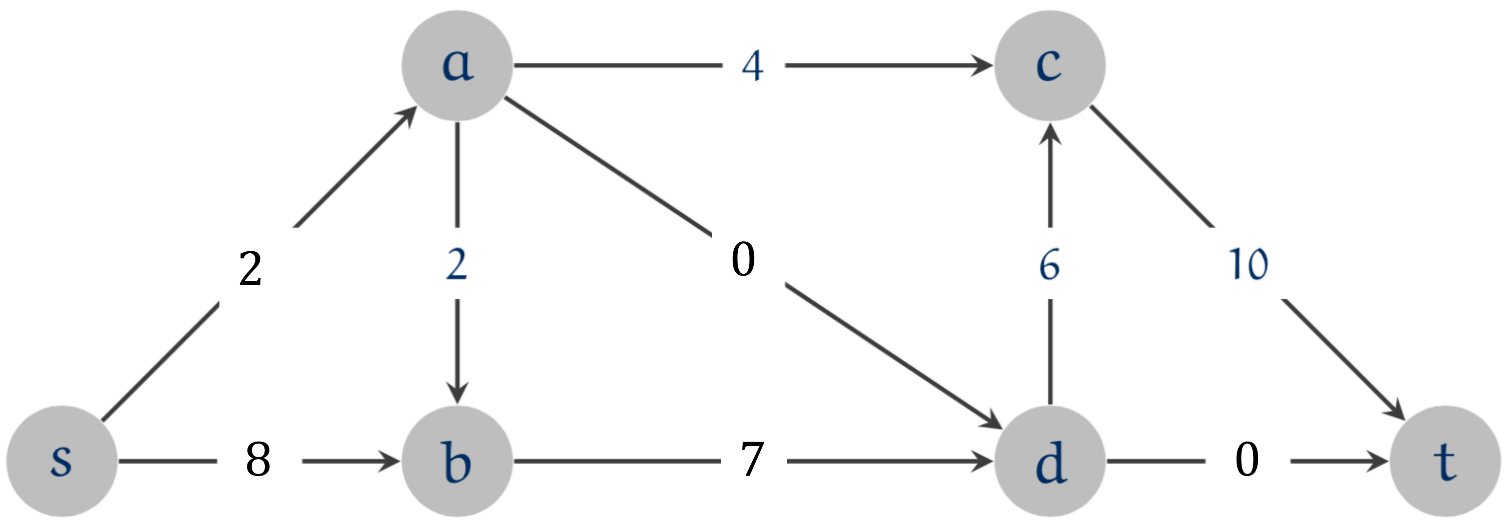
$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8$

Demo: Simple Greedy Fails



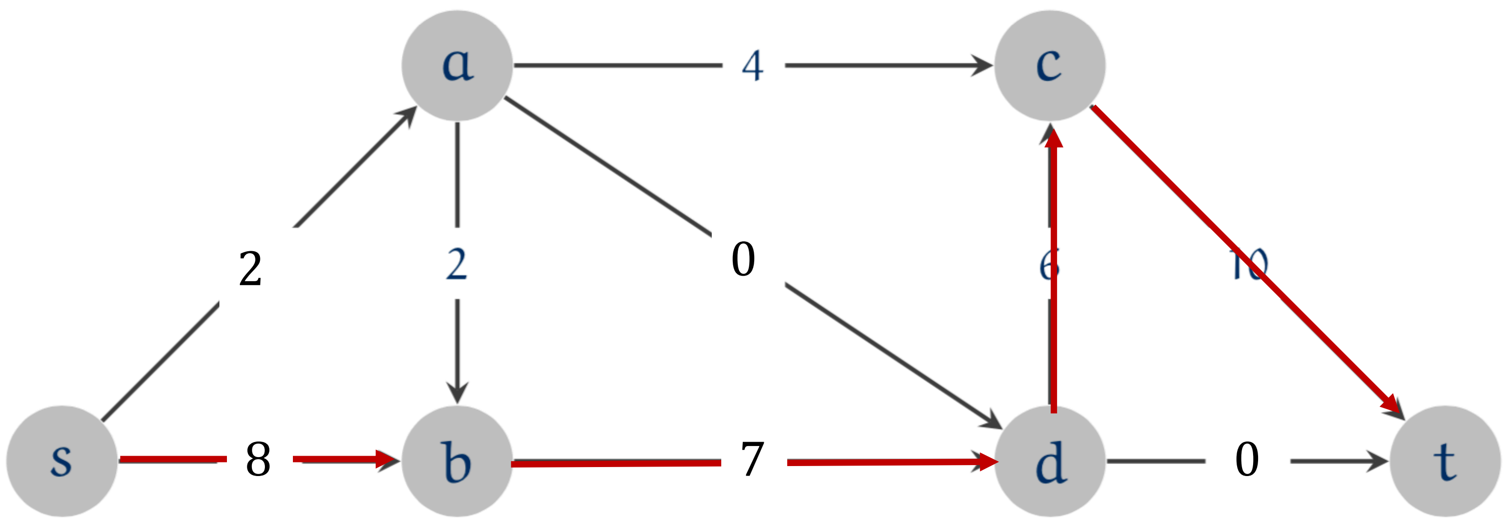
$$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$$

Demo: Simple Greedy Fails



$$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$$

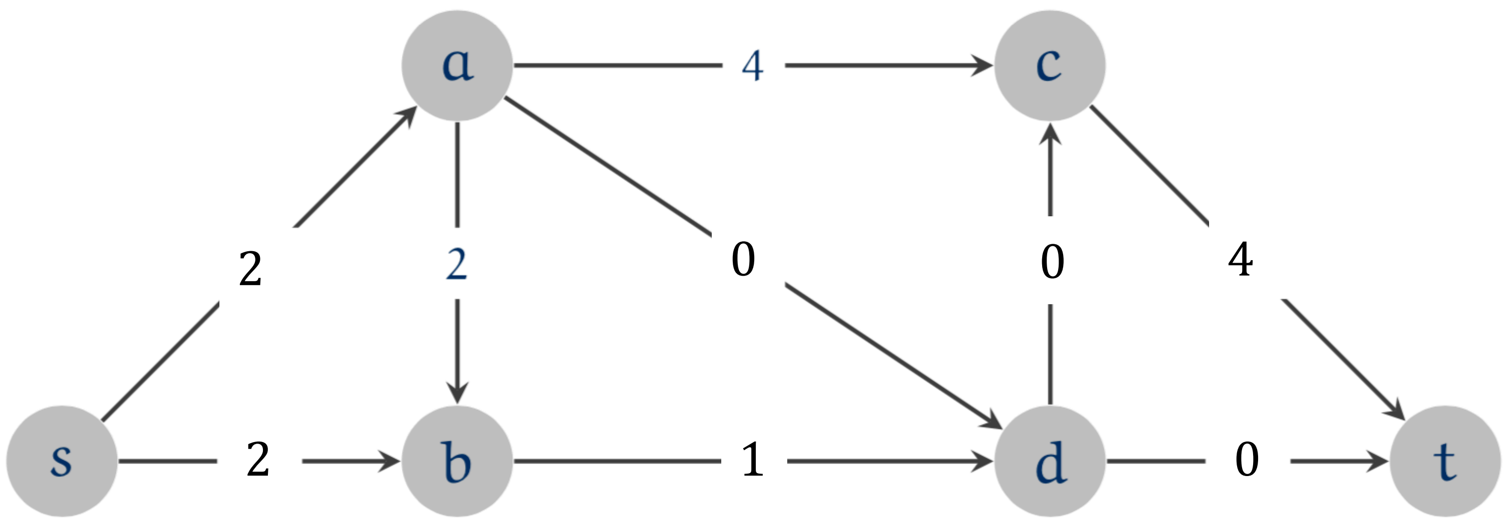
Demo: Simple Greedy Fails



$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$

$\rightarrow Val(f) = 10 + 6 = 16$

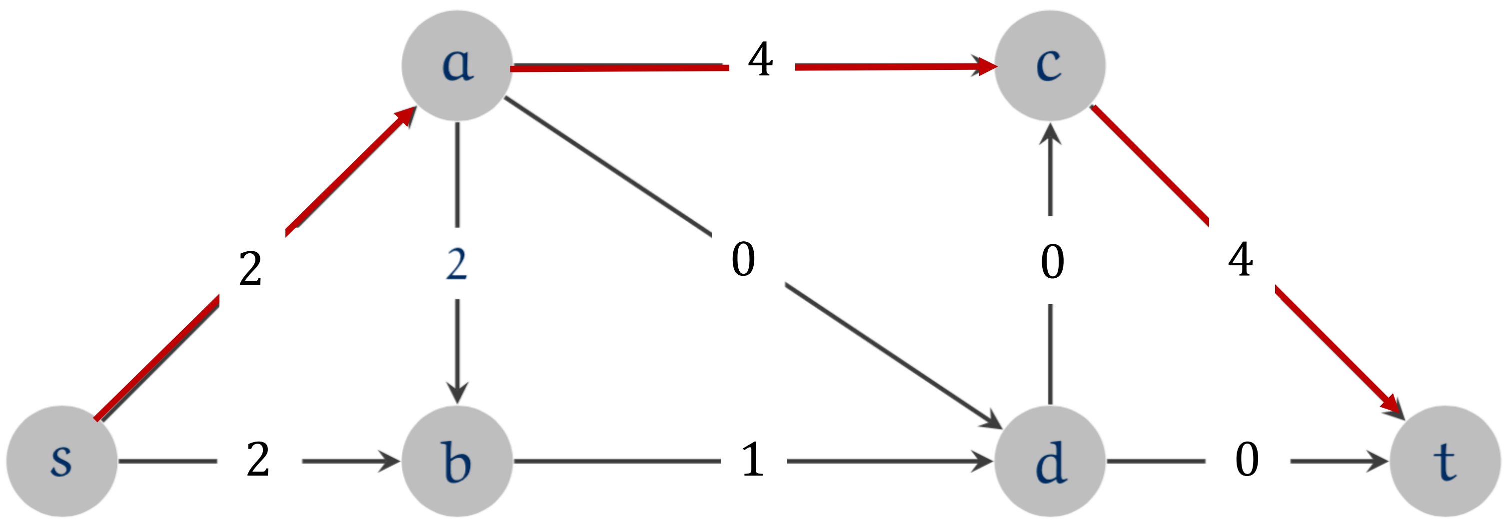
Demo: Simple Greedy Fails



$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$

$\rightarrow Val(f) = 10 + 6 = 16$

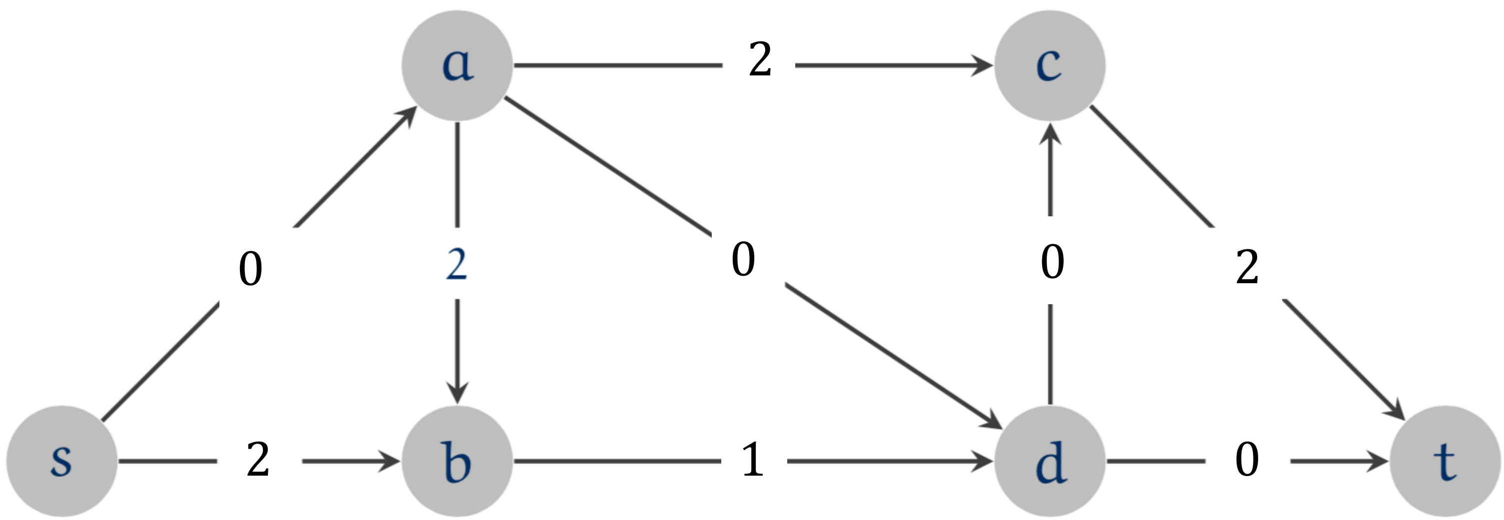
Demo: Simple Greedy Fails



$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$

$\rightarrow Val(f) = 10 + 6 = 16 \quad \rightarrow Val(f) = 16 + 2 = 18$

Demo: Simple Greedy Fails



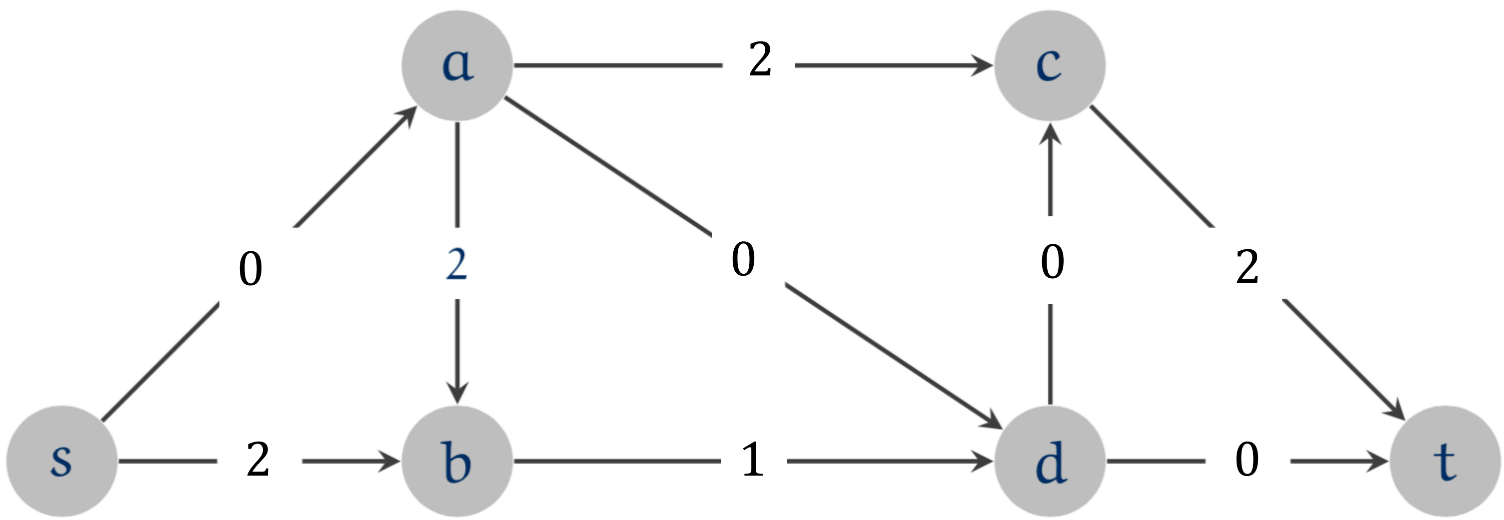
$Val(f) = 0 \quad \rightarrow Val(f) = 0 + 8 = 8 \quad \rightarrow Val(f) = 8 + 2 = 10$

$\rightarrow Val(f) = 10 + 6 = 16 \quad \rightarrow Val(f) = 16 + 2 = 18$

But optimal $Val(f) = 19$!

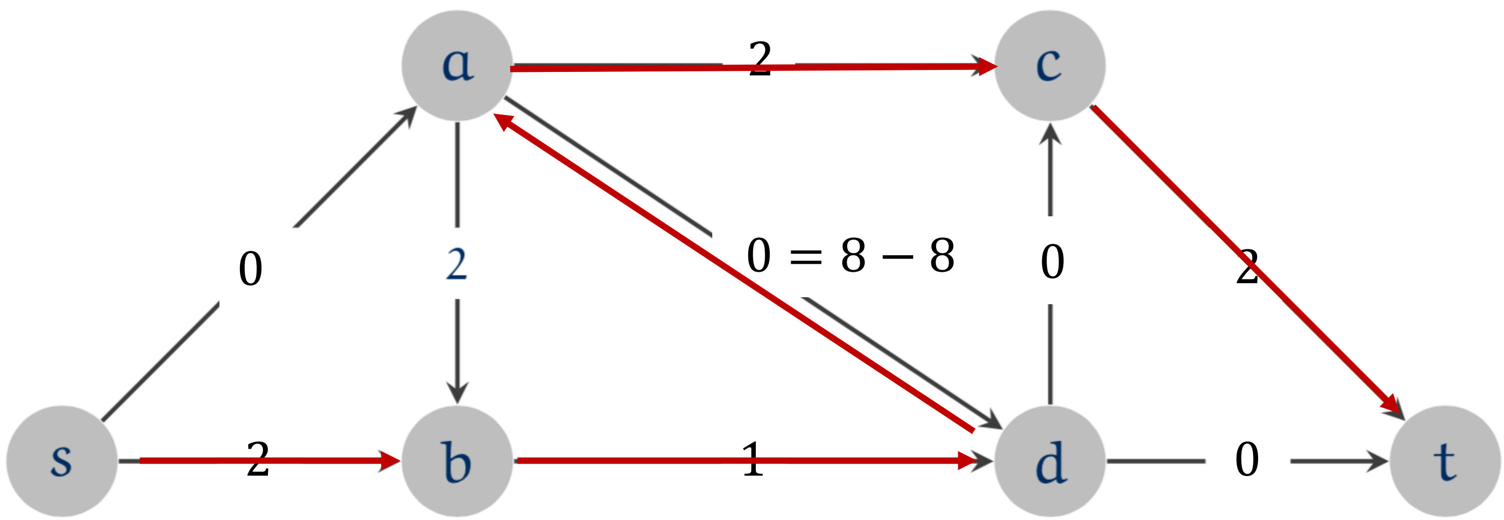
What goes wrong?

- Only allow increasing flow amount on an edge previously
- Need to allow reducing the flow amount as well



What goes wrong?

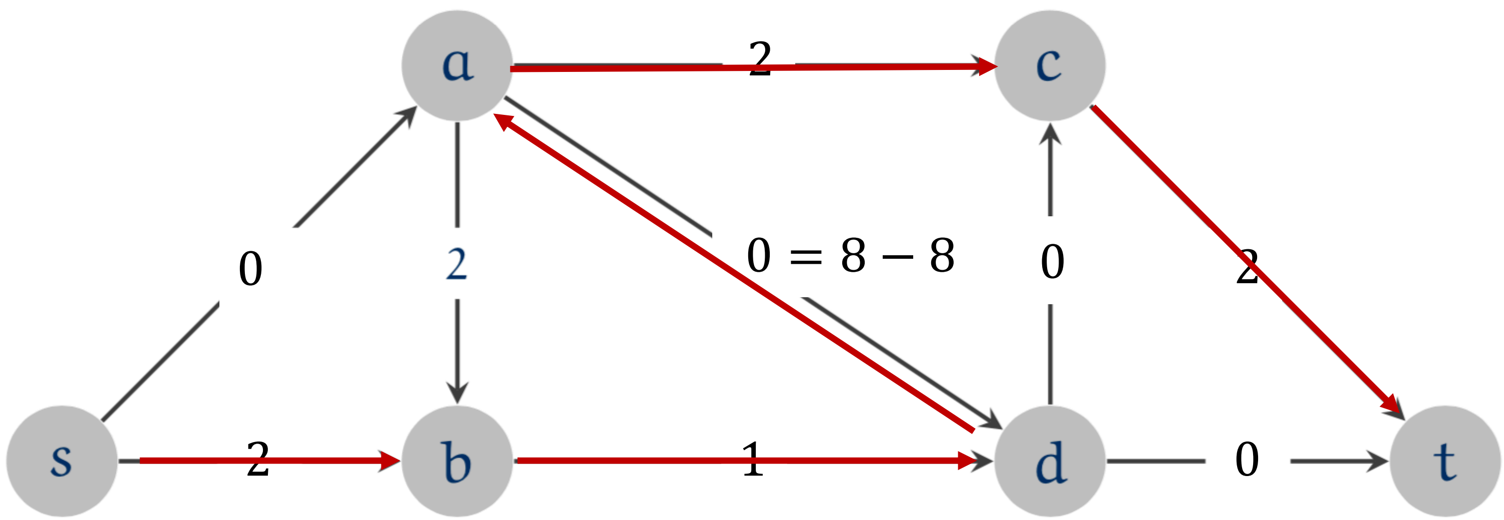
- Only allow increasing flow amount on an edge previously
- Need to allow reducing the flow amount as well



The missing one unit flow is here!

What goes wrong?

- Only allow increasing flow amount on an edge previously
- Need to allow reducing the flow amount as well



We have 8 units from $a \rightarrow d$, should be allowed to flow 1 unit back from $d \rightarrow a$

- Or equivalently, flow only 7 units from $a \rightarrow d$
- That is, Flow with two different directions cancels out!

The Formal Way: Residual Graph

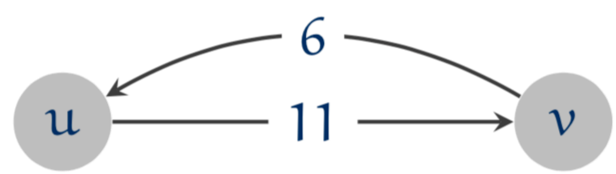
Original edge: $e = (u, v) \in E$

- Flow $f(e)$
- Capacity $c(e)$

Residual edge: $e = (u, v) \in E$

- "undo" flow sent
- $e = (u, v)$ and $e^R = (v, u)$
- Residual capacity

$$\begin{cases} c_f(e) = c(e) - f(e) \\ c_f(e^R) = f(e) \end{cases}$$



The Formal Way: Residual Graph

Original edge: $e = (u, v) \in E$

- Flow $f(e)$
- Capacity $c(e)$

Residual edge: $e = (u, v) \in E$

- "undo" flow sent
- $e = (u, v)$ and $e^R = (v, u)$
- Residual capacity

$$\begin{cases} c_f(e) = c(e) - f(e) \\ c_f(e^R) = f(e) \end{cases}$$

- We assumed no parallel edges, i.e., if $e \in E$ then $e^R \notin E$
- But residual edges easily generalize to parallel edges
  - Suppose $e^R \in E$ has capacity $c(e^R)$ and flow $f(e^R)$

$$\begin{cases} c_f(e) = [c(e) - f(e)] + f(e^R) \\ c_f(e^R) = f(e) + [c(e^R) - f(e^R)] \end{cases}$$

The Formal Way: Residual Graph

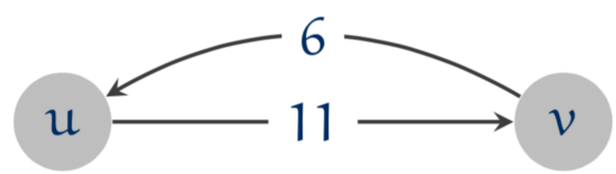
Original edge: $e = (u, v) \in E$

- Flow $f(e)$
- Capacity $c(e)$

Residual edge: $e = (u, v) \in E$

- "undo" flow sent
- $e = (u, v)$ and $e^R = (v, u)$
- Residual capacity

$$\begin{cases} c_f(e) = c(e) - f(e) \\ c_f(e^R) = f(e) \end{cases}$$



Residual Graph: $G_f = (V, E_f)$

- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$
- Residual edges have positive residual capacity, and allow us to "undo" flow
- Key property: if f' is a flow in G_f, then $f + f'$ is a flow in G

Augmenting Path

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

Algorithm: $\mathrm{Augment}(f, c, P)$:

$b =$ bottleneck capacity of path P;
foreach *edge* $e \in P$ **do**
  if $e \in E$ **then**
    $f(e) = f(e) + b$;
  else
    $f(e^R) = f(e^R) - b$;
return f

Key Property: Let f be a flow and let P be an augmenting path in G_f. Then new flow f' satisfies $val(f') = val(f) + bottleneck(G_f, P)$

Ford-Fulkerson Algorithm

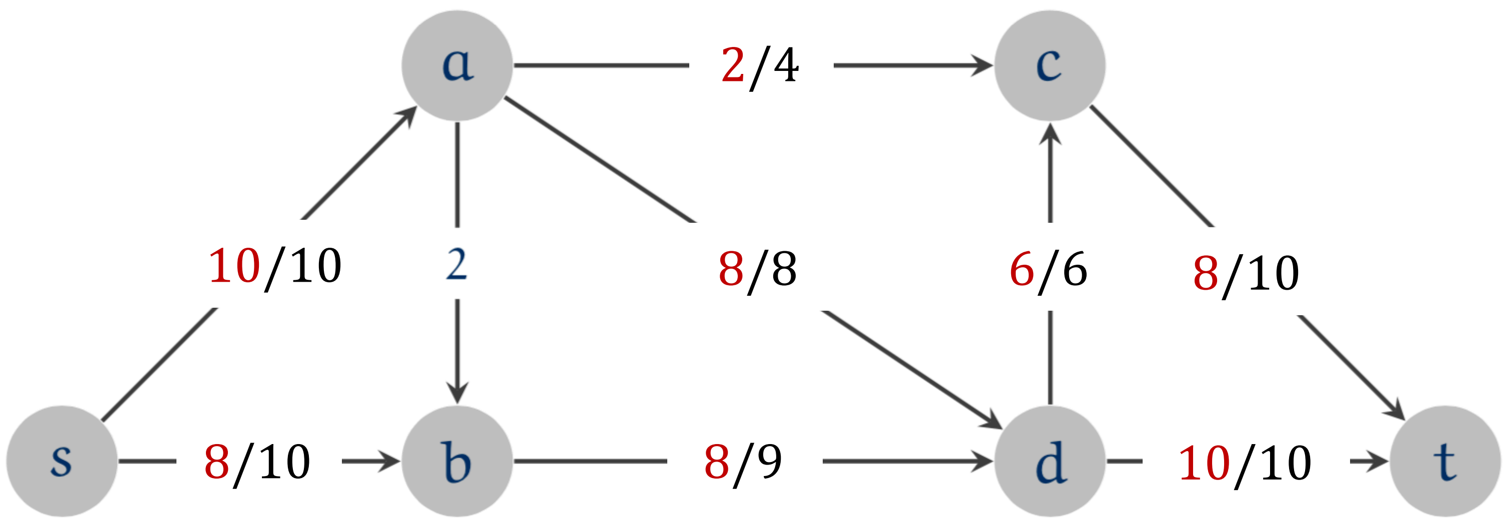
Greedy algorithm using augmenting path

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an augmenting $s - t$ path P in the residual graph G_f
- Augment flow along path P
- Repeat, until stuck

Algorithm: Ford-Fulkerson(G, s, t, c):

```
foreach edge e ∈ E do
    f(e) = 0;
G_f = residual graph;
while there exists an augmenting path P in G_f do
    f = Augment(f, c, P);
    Update G_f;
return f
```

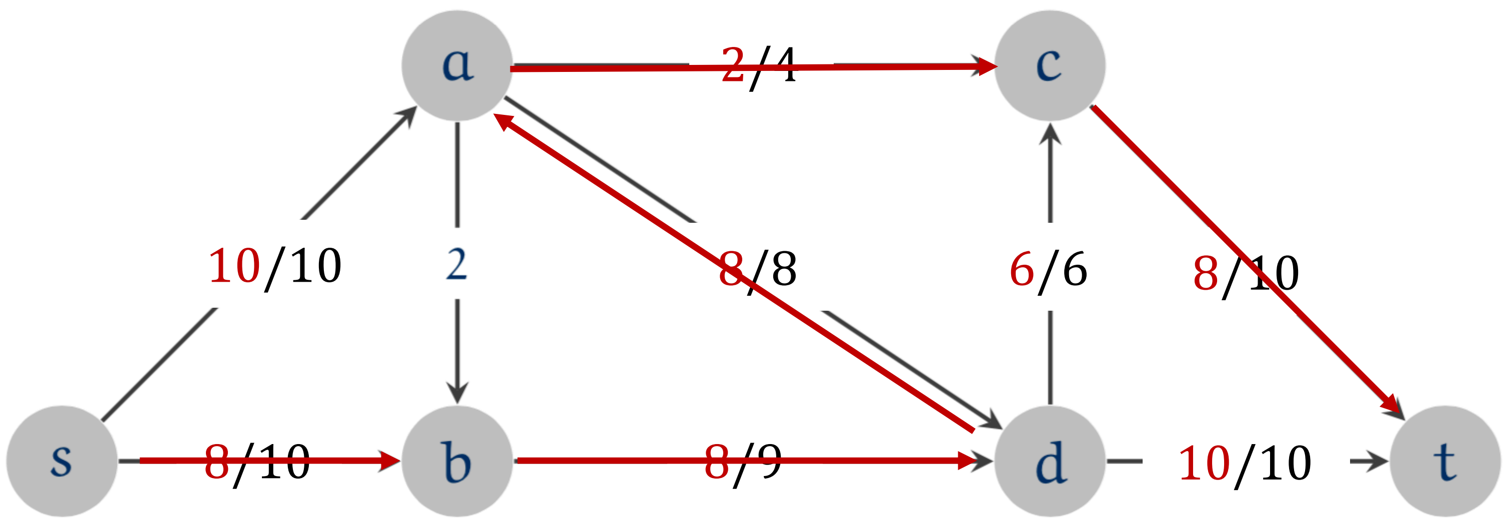
Demo: Simple Greedy Fails



$Val(f) = 18$

But optimal $Val(f) = 19$!

Demo: How FF Saved It

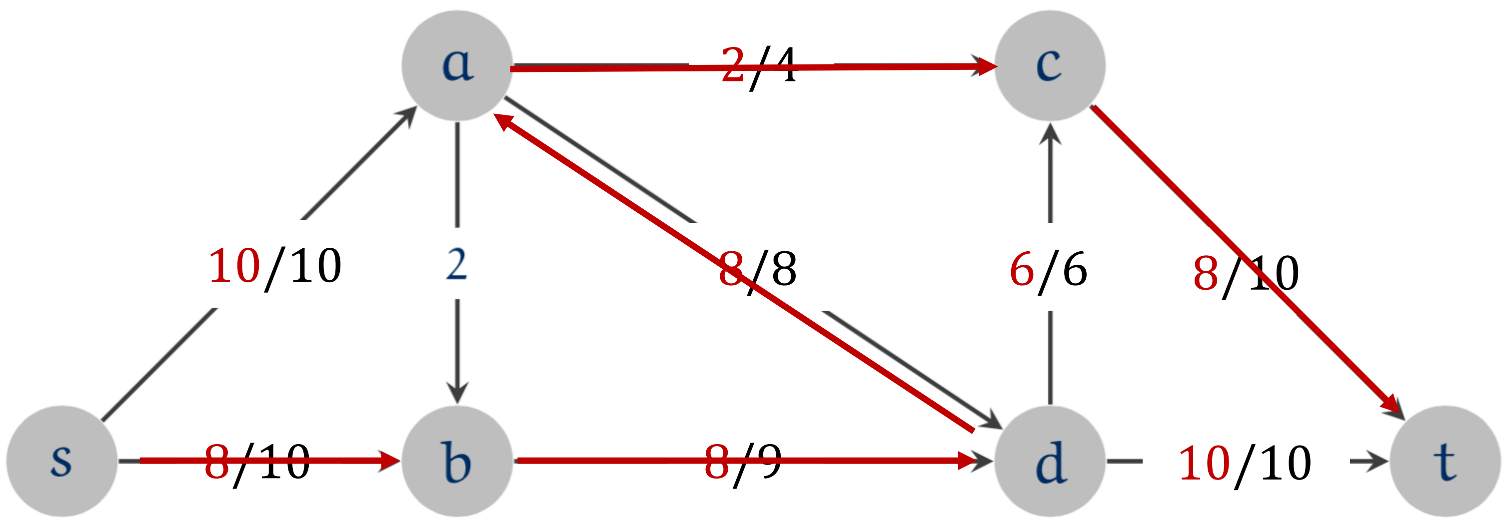


$Val(f) = 18$

An augment path with bottleneck 1

But optimal $Val(f) = 19$!

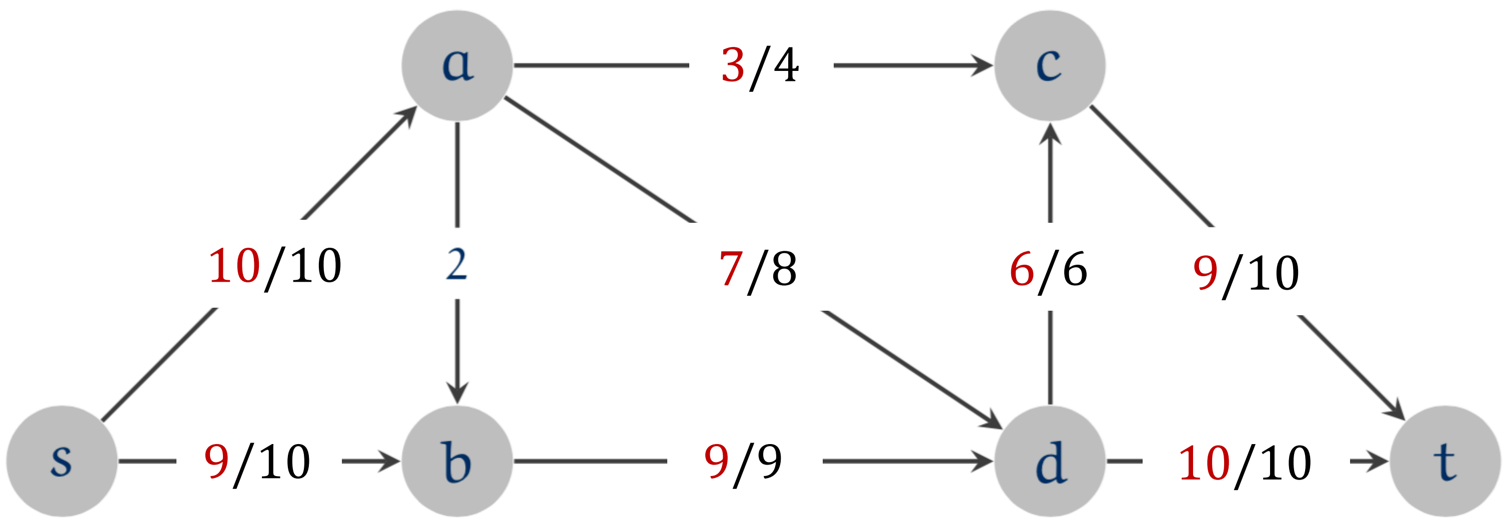
Demo: How FF Saved It



$Val(f) = 18 \quad \rightarrow Val(f) = 18 + 1 = 19$

But optimal $Val(f) = 19$!

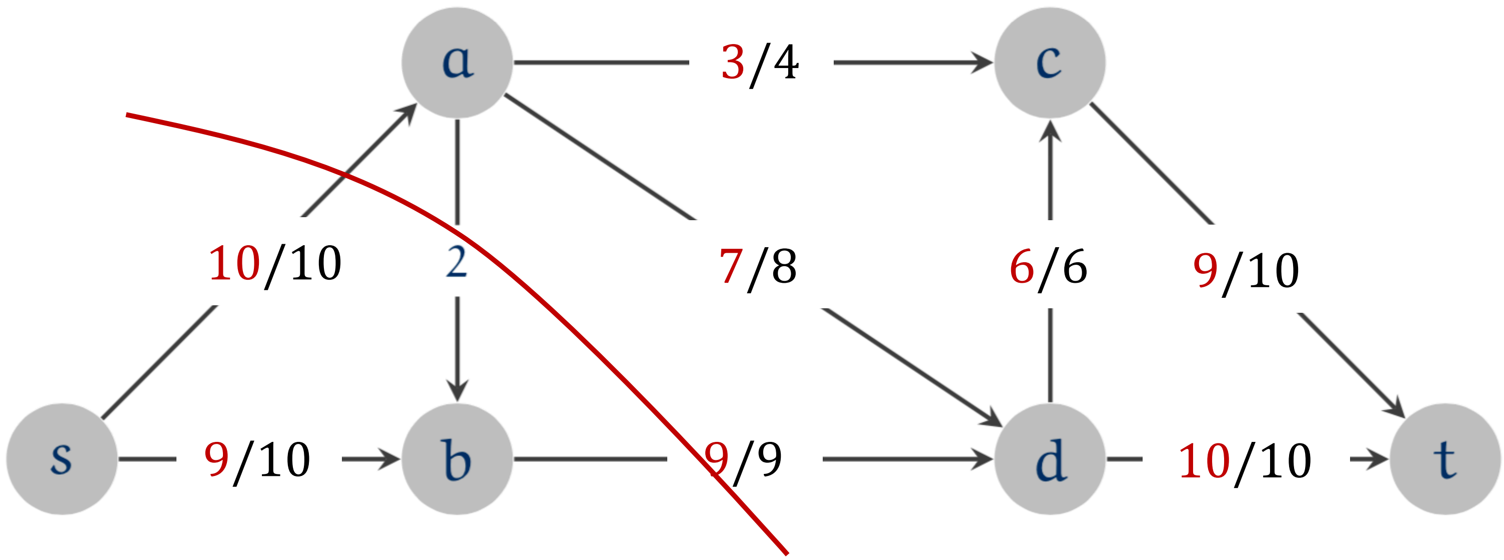
Demo: How FF Saved It



$Val(f) = 18 \quad \rightarrow Val(f) = 18 + 1 = 19$

But optimal $Val(f) = 19$!

Demo: How FF Saved It



This cut's capacity is 19

Can you easily see why 19 is optimal?

Relationship Between Flows and Cuts

Flow Value Lemma: Let f be any flow and (A, B) be any cut. Then the net flow from A to B equals the value of f. That is

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = val(f)$$

Proof

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

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

By flow conservation

Relationship Between Flows and Cuts

Flow Value Lemma: Let f be any flow and (A, B) be any cut. Then the net flow from A to B equals the value of f. That is

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = val(f)$$

Proof

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

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

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

$f(e)$ is canceled out, except for those existing or entering A

Weak Duality

Weak Duality Lemma: Let f be any flow and (A, B) be any cut. Then $\mathrm{val}(f) \leq cap(A, B)$.

- Therefore, we know our flow was optimal in previous Demo
- Proof: using previous lemma

$$val(f) = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ into\ A} f(e)$$

$$\leq \sum_{e\ out\ of\ A} f(e)$$

$$\leq \sum_{e\ out\ of\ A} c(e) = cap(A, B)$$

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu