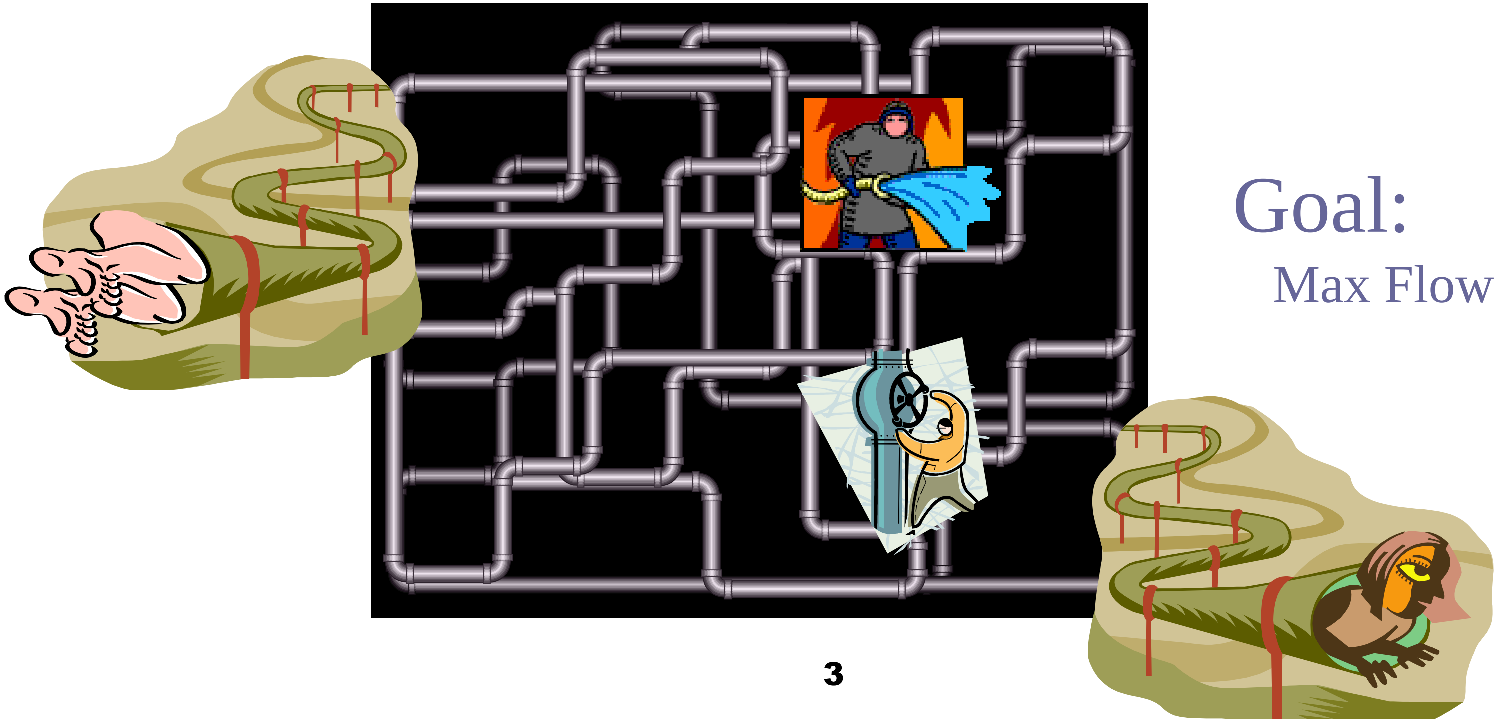
# Flow Networks

# Types of Networks

- Internet
- Telephone
- Cell
- Highways
- Rail
- Electrical Power
- Water
- Sewer
- Gas
- …

# Network Flow

- A Network is a directed graph $G$
- Edges represent pipes that carry flow
- Each edge $(u,v)$ has a maximum capacity $c(u,v)$
- A source node $s$ in which flow arrives
- A sink node $t$ out which flow leaves

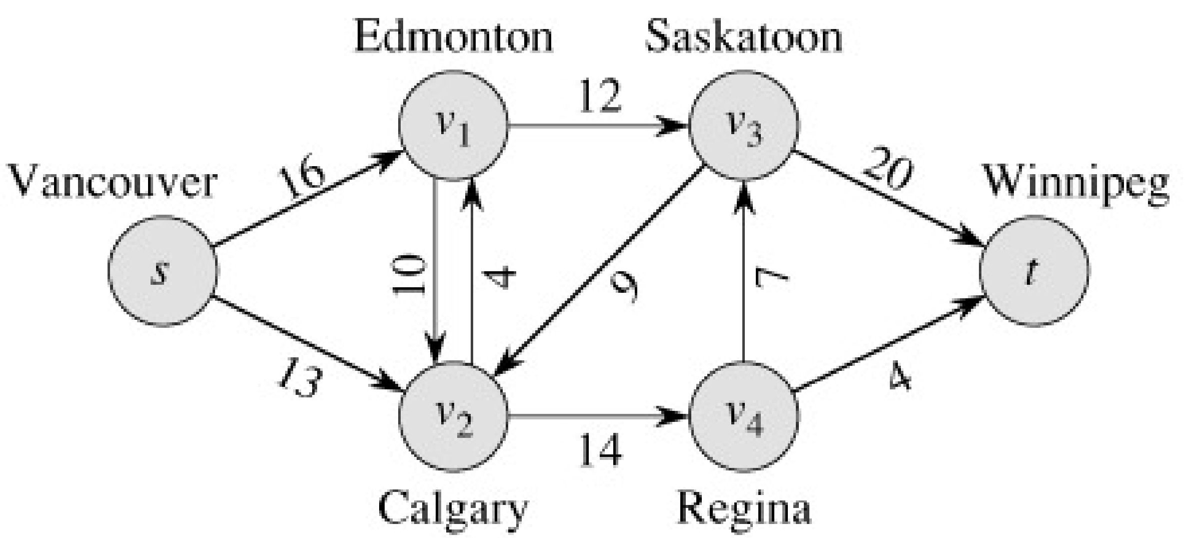


Goal:

Max Flow

# The Problem

- Use a graph to model material that flows through conduits.
- Each edge represents one conduit, and has a capacity, which is an upper bound on the flow rate, in units/time.
- Can think of edges as pipes of different sizes.
- Want to compute max rate that we can ship material from a designated source to a designated sink.

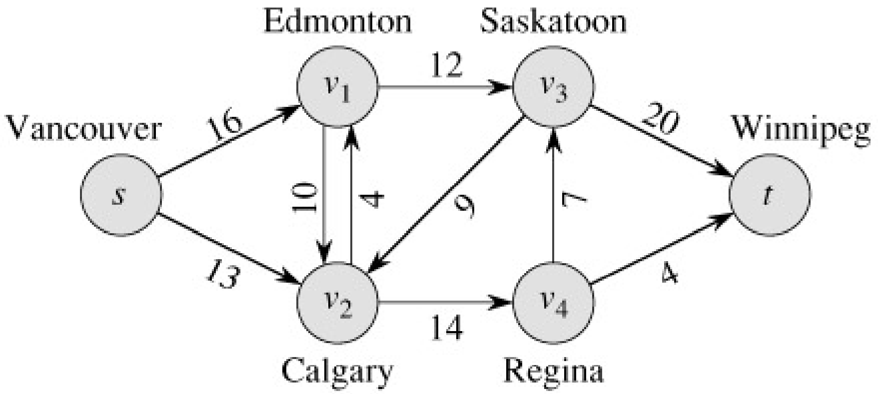


# What is a Flow Network?

- Each edge (u,v) has a nonnegative capacity c(u,v).
- If (u,v) is not in E, assume c(u,v)=0.

  e.g., $c(s,v_1)=16$; $c(v_1,s)=0$; $c(v_2,v_3)=0$

- We have a source s, and a sink t.
- Assume that every vertex v in V is on some path from s to t.



# What is a Flow in a Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

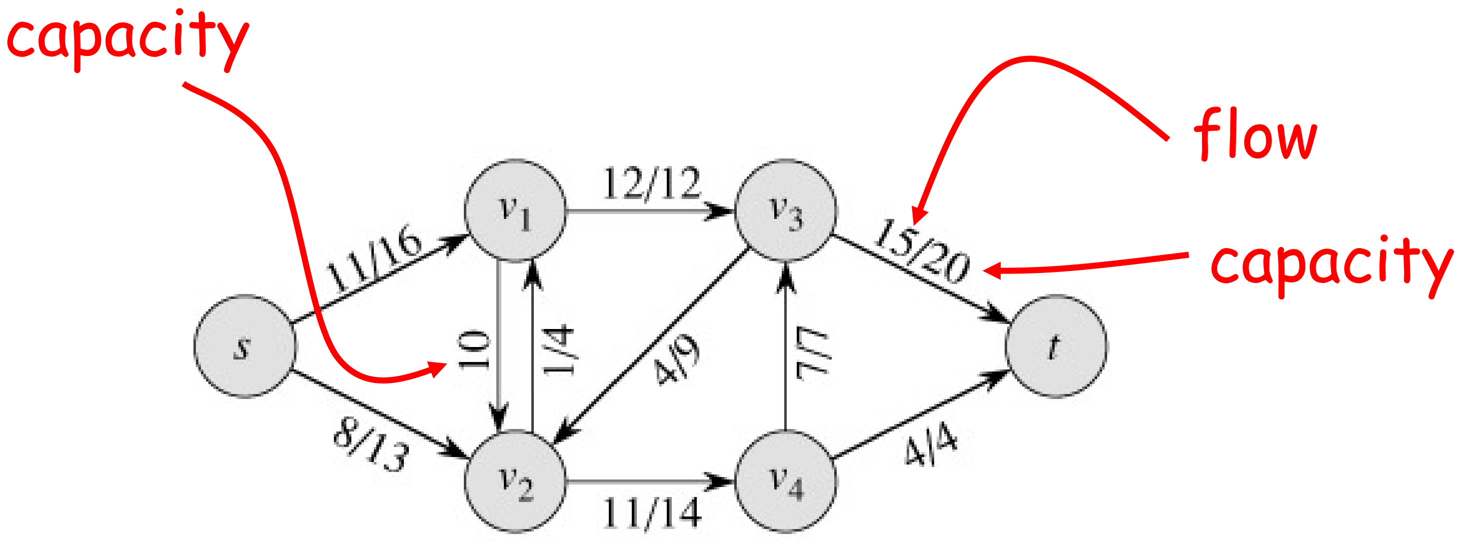
Capacity constraint: $\forall u,v \in V,\ f(u,v) \le c(u,v)$

Skew symmetry: $\forall u,v \in V,\ f(u,v) = -f(v,u)$

Flow conservation: $\forall u \in V - \{s,t\},\ \sum_{v \in V} f(u,v) = 0$

- Notes:
  - The skew symmetry condition implies that f(u,u)=0.
  - We show only the *positive* capacity/flows in the flow network.

# Example of a Flow:




- $f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$.
- $f(v_1, v_2) = -1$ , $c(v_1, v_2) = 10$
- $f(v_3, s) + f(v_3, v_1) + f(v_3, v_2) + f(v_3, v_4) + f(v_3, t) =$

  $0 \quad + \quad (-12) \quad + \quad 4 \quad + \quad (-7) \quad + \quad 15 \quad = \quad 0$

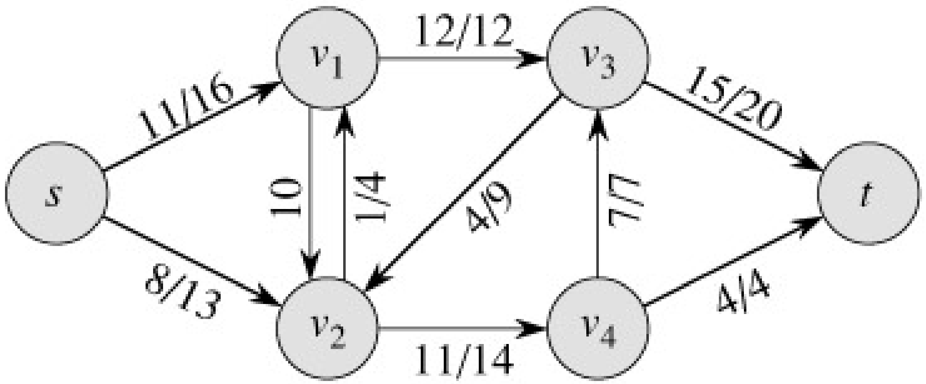
# The Value of a flow

- The value of a flow is given by

$$|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$$

  - This is the total flow leaving s = the total flow arriving in t.

# Example:




|f| (flow leaving 's' ) = $f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t)$

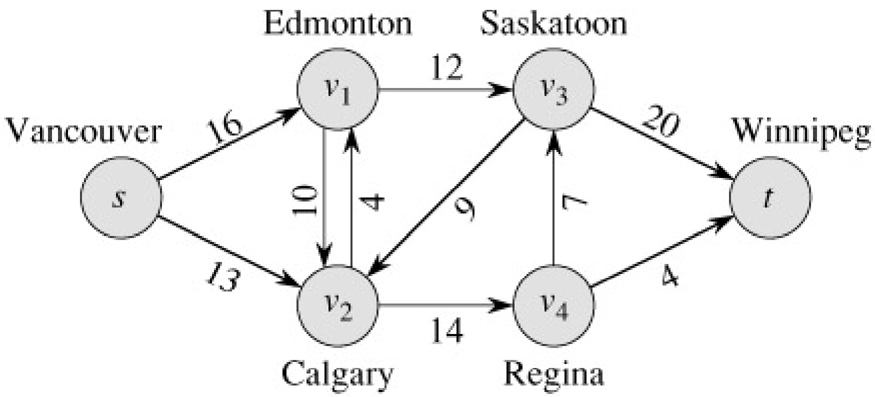
**11** + **8** + **0** + **0** + **0** = **19**

|f| (flow arriving 't' ) = $f(s, t) + f(v_1, t) + f(v_2, t) + f(v_3, t) + f(v_4, t)$

**0** + **0** + **0** + **15** + **4** = **19**

# A flow in a network

- We assume that there is only flow in one direction at a time.




- Sending 7 trucks from Edmonton to Calgary and 3 trucks from Calgary to Edmonton has the same net effect as sending 4 trucks from Edmonton to Calgary.

# Residual Networks

- The residual capacity of an edge *(u,v)* in a network with a flow $f$ is given by:

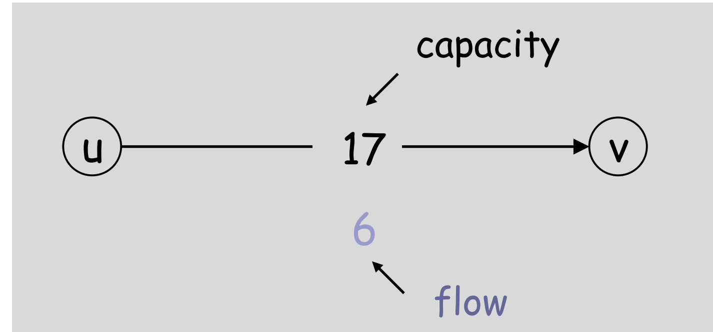
$$\boxed{c_f(u,v) = c(u,v) - f(u,v)}$$

- The residual network of a graph *G* induced by a flow *f* is the graph including only the edges with positive residual capacity, i.e.,

$$G_f = (V, E_f), \text{ where } E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

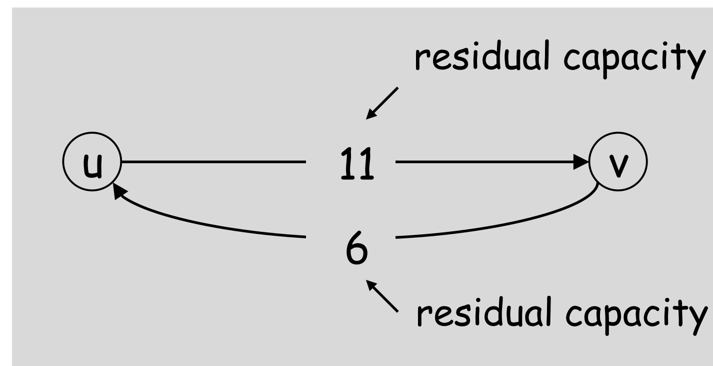
# Residual Graph/Network

- 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:

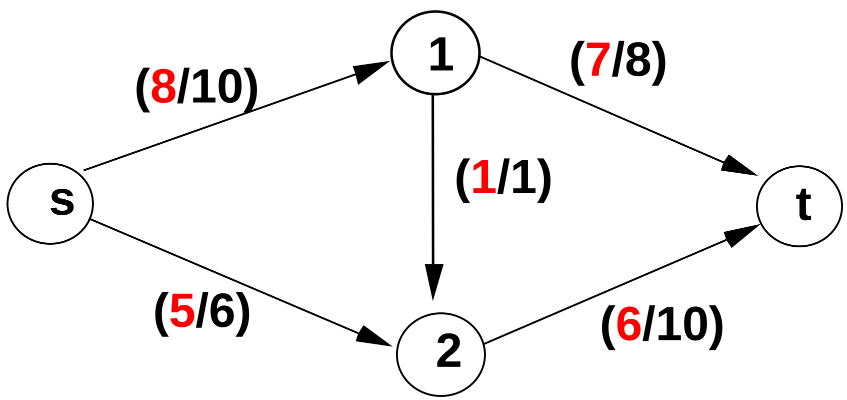
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

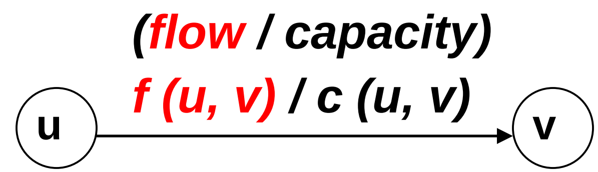



- Residual graph: $G_f = (V, E_f)$.
  - Residual edges with positive residual capacity.
  - $E_f = \{e : f(e) < c(e)\} \cup \{e^R : c(e) > 0\}$.

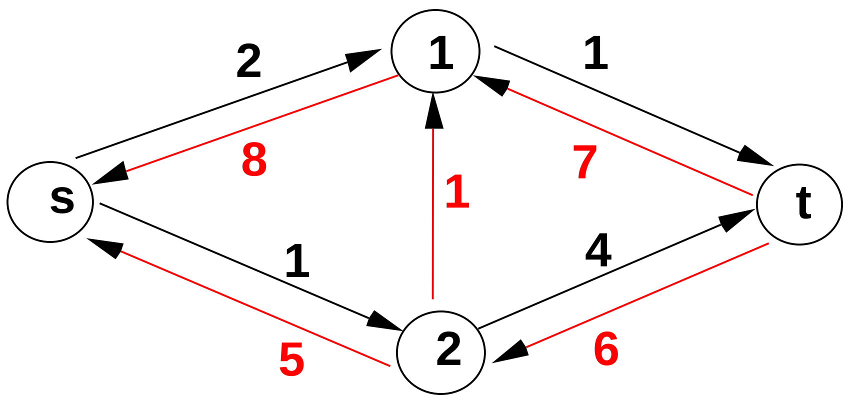
# The Residual Network

**Flow Network**

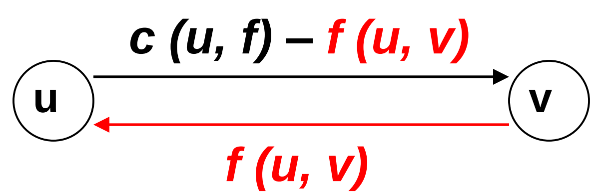



**Residual Network**

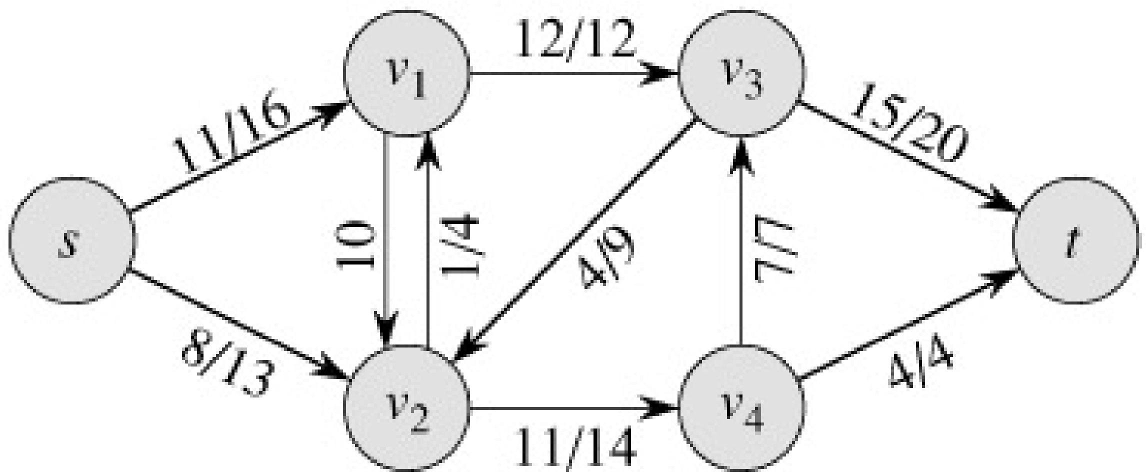



***Residual capacity r (u ,v)***

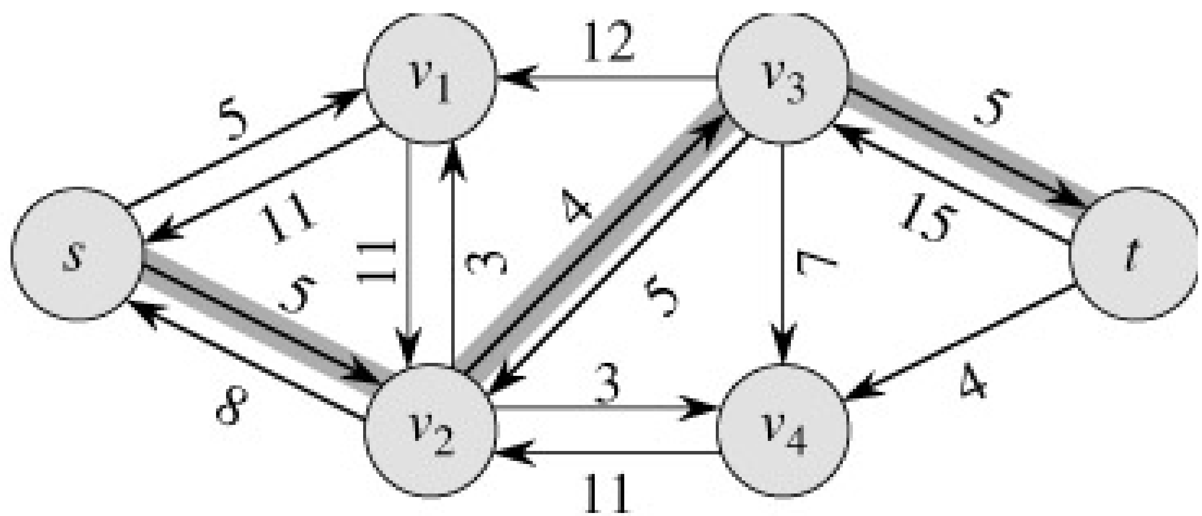


# Example of Residual Network

Flow Network:




Residual Network:



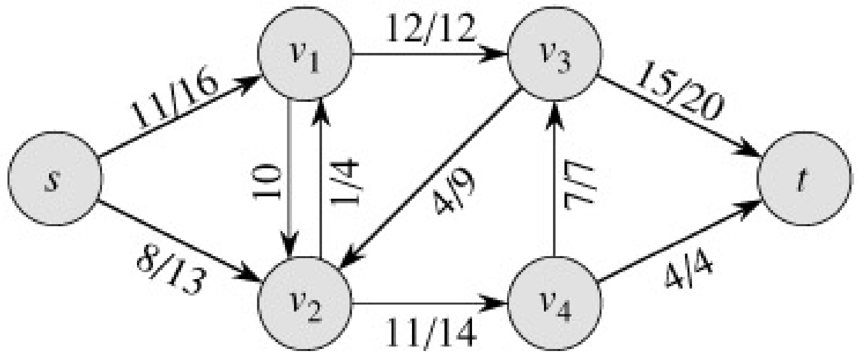
# Augmenting Path

- An augmenting path *p* is a simple path from s to t on the residual network.
- We can put more flow from *s* to *t* through *p*.
- We call the maximum capacity, by which we can increase the flow on *p,* the residual capacity of *p*.

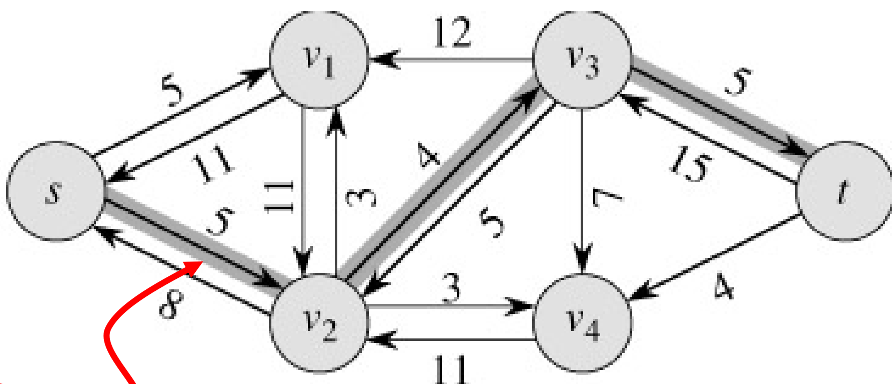
$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\}$$

# Augmenting Paths

Network:




Residual Network:




Augmenting path

The residual capacity of this augmenting path is 4.

# Ford-Fulkerson algorithm

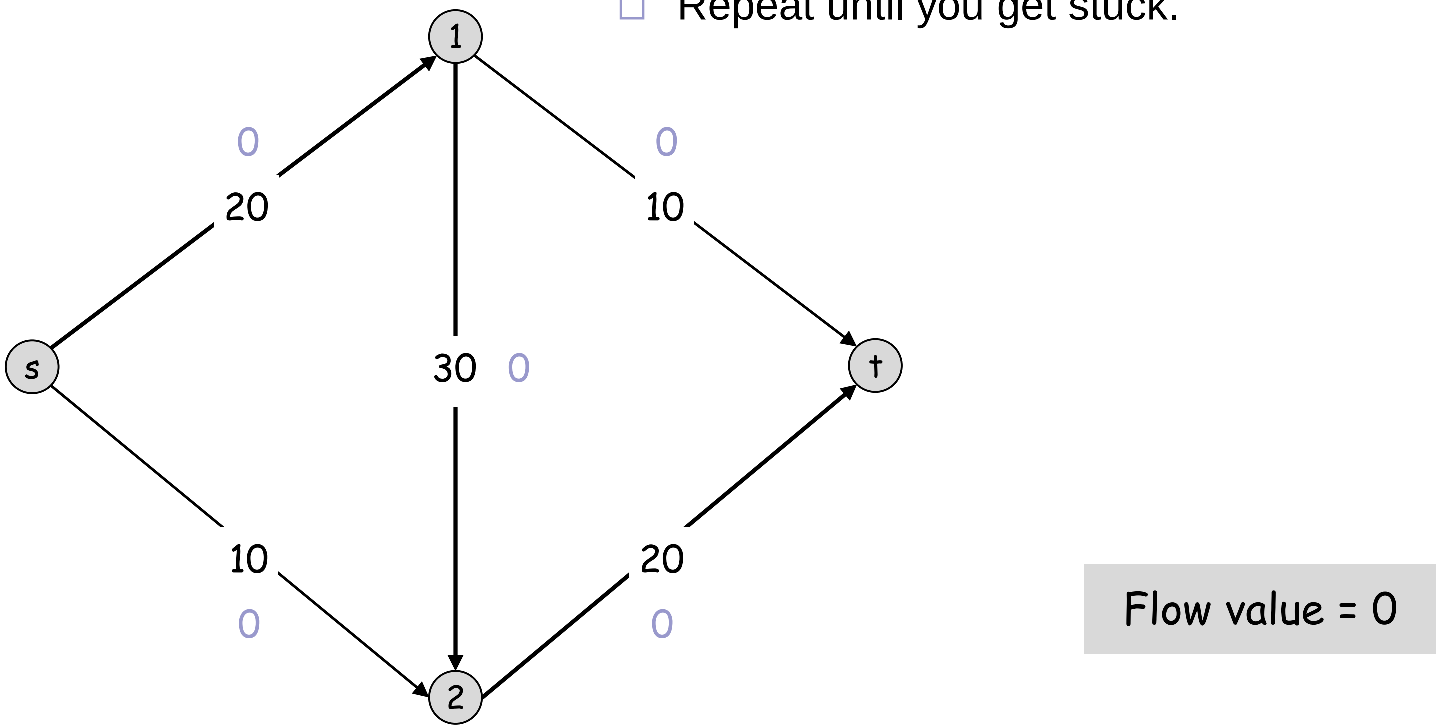
Start with 0 flow.

While there exists an augmenting path:

- find an augmenting path
- compute bottleneck capacity
- increase flow on that path by bottleneck capacity

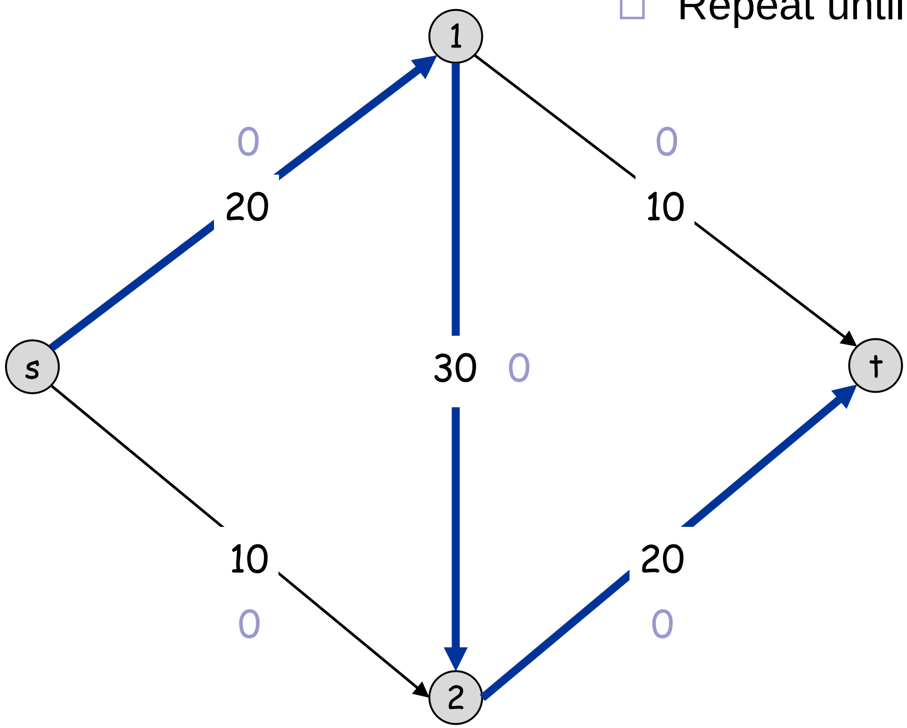
# Ford-Fulkerson algorithm

- Greedy algorithm.
  - Start with f(e) = 0 for all edge e ∈ 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.



# Ford-Fulkerson algorithm

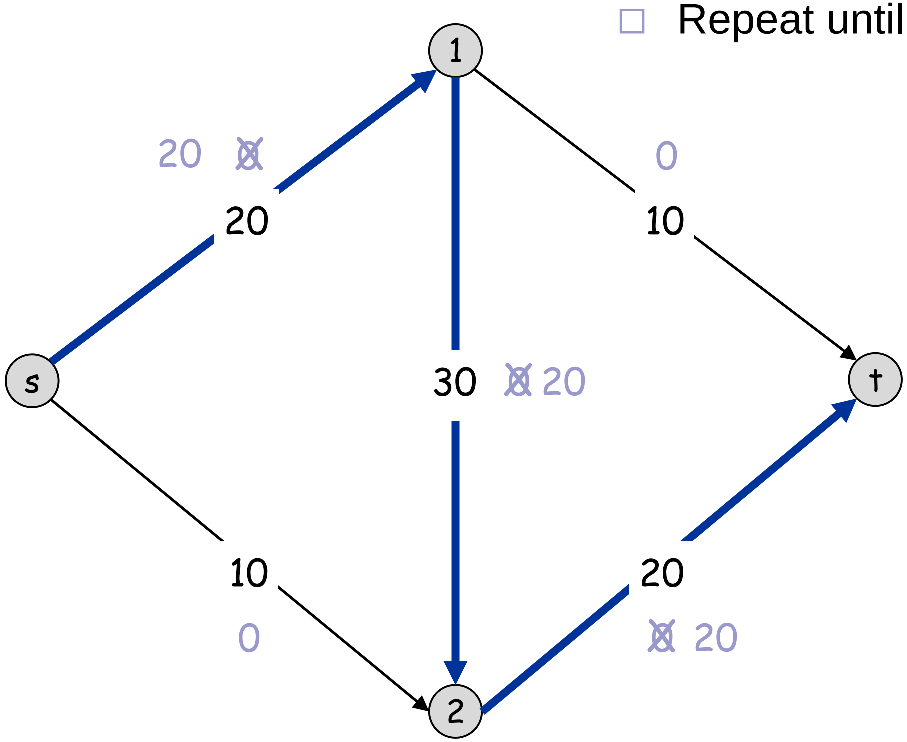
- Greedy algorithm.
  - Start with f(e) = 0 for all edge e ∈ E.
  - Find an s-t path P where each edge has f(e) < c(e).
  - Augment flow along path P.
  - Repeat until you get stuck.




Flow value = 0

# Ford-Fulkerson algorithm

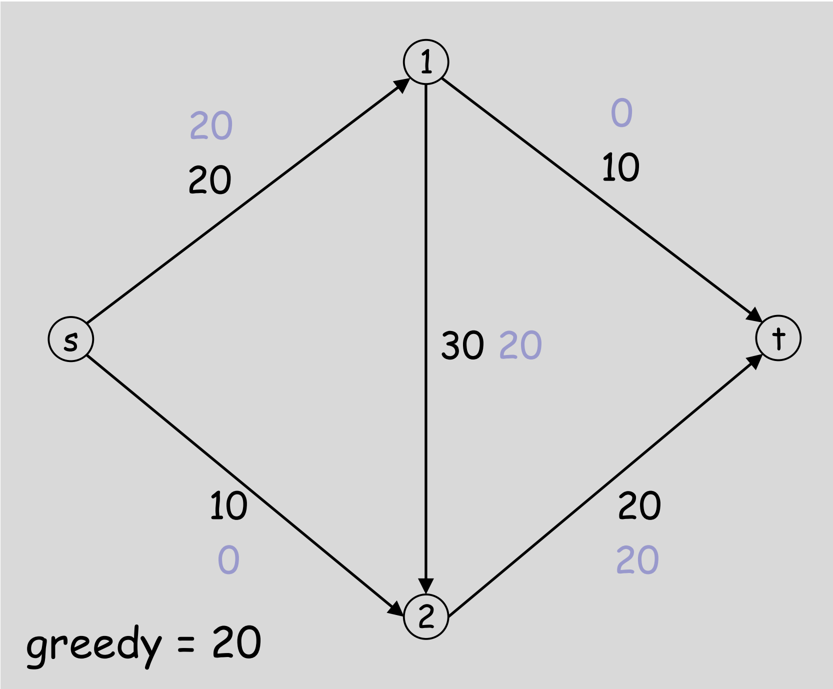
- Greedy algorithm.
  - Start with f(e) = 0 for all edge e $\in$ E.
  - Find an s-t path P where each edge has f(e) < c(e).
  - Augment flow along path P.
  - Repeat until you get stuck.

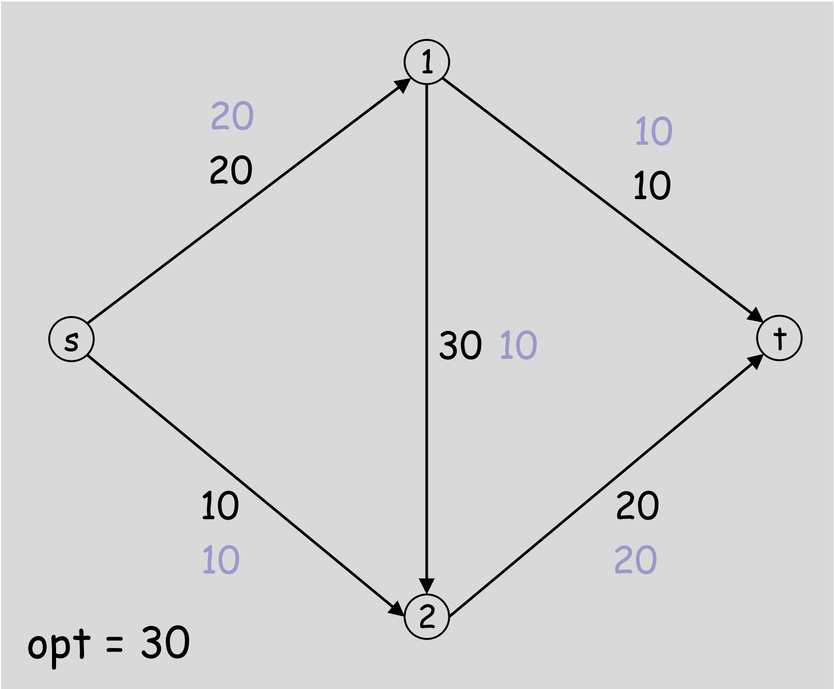



Flow value = 20

# Towards a Max Flow Algorithm

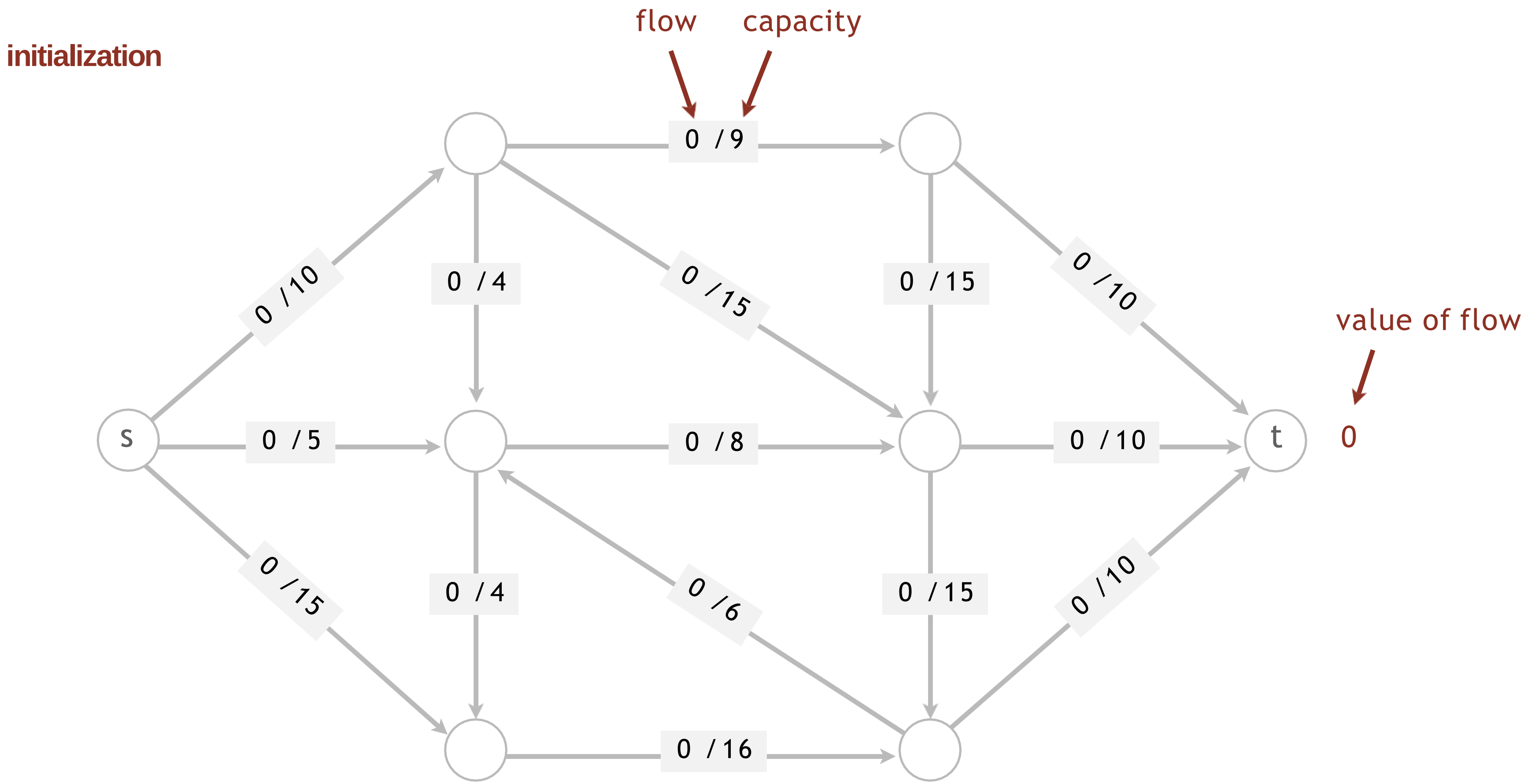
- Greedy algorithm.
  - Start with f(e) = 0 for all edge e $\in$ E.
  - Find an s-t path P where each edge has f(e) < c(e).
  - Augment flow along path P.
  - Repeat until you get stuck.



# Ford-Fulkerson algorithm

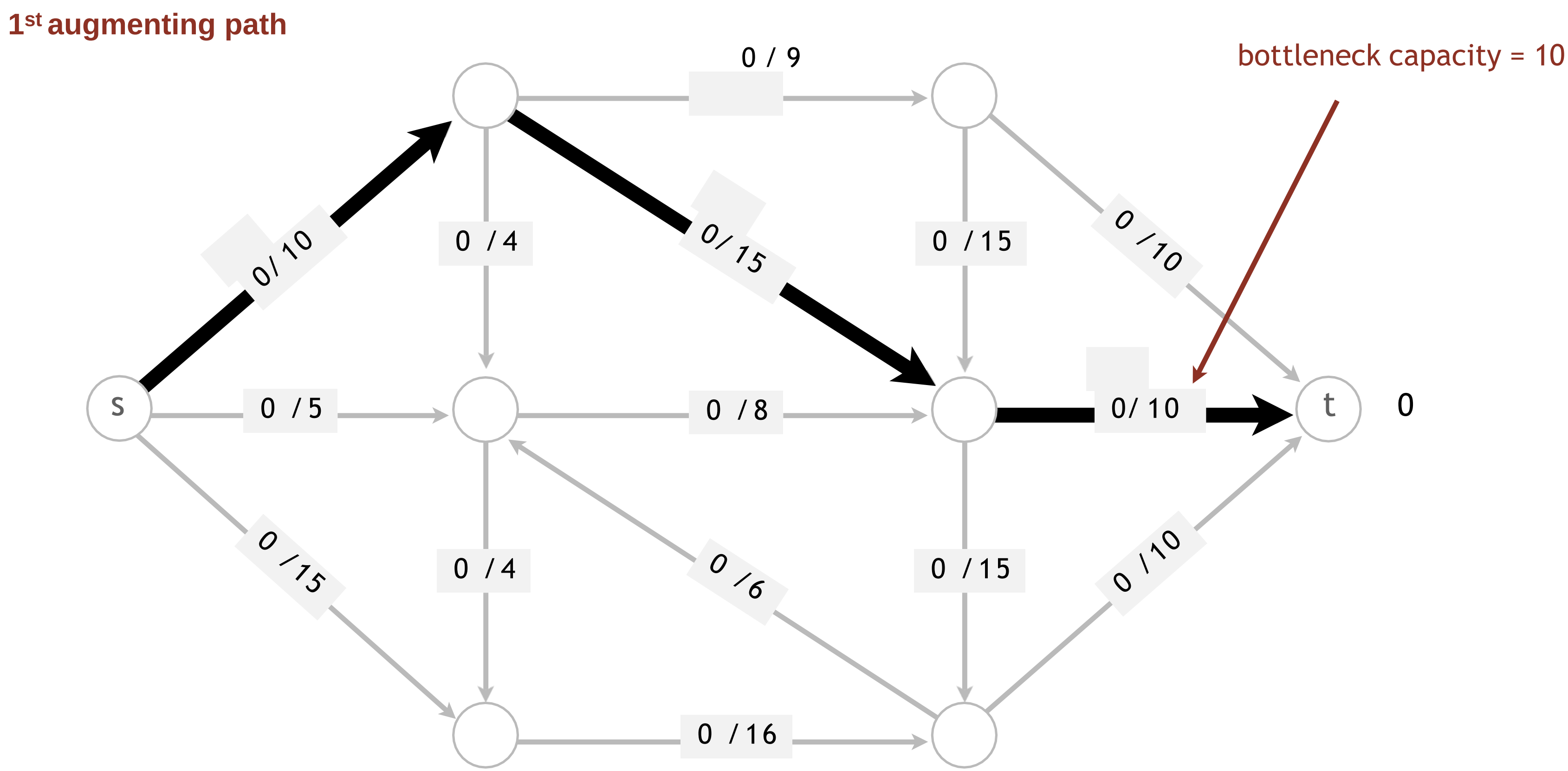
Initialization. Start with 0 flow.



# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

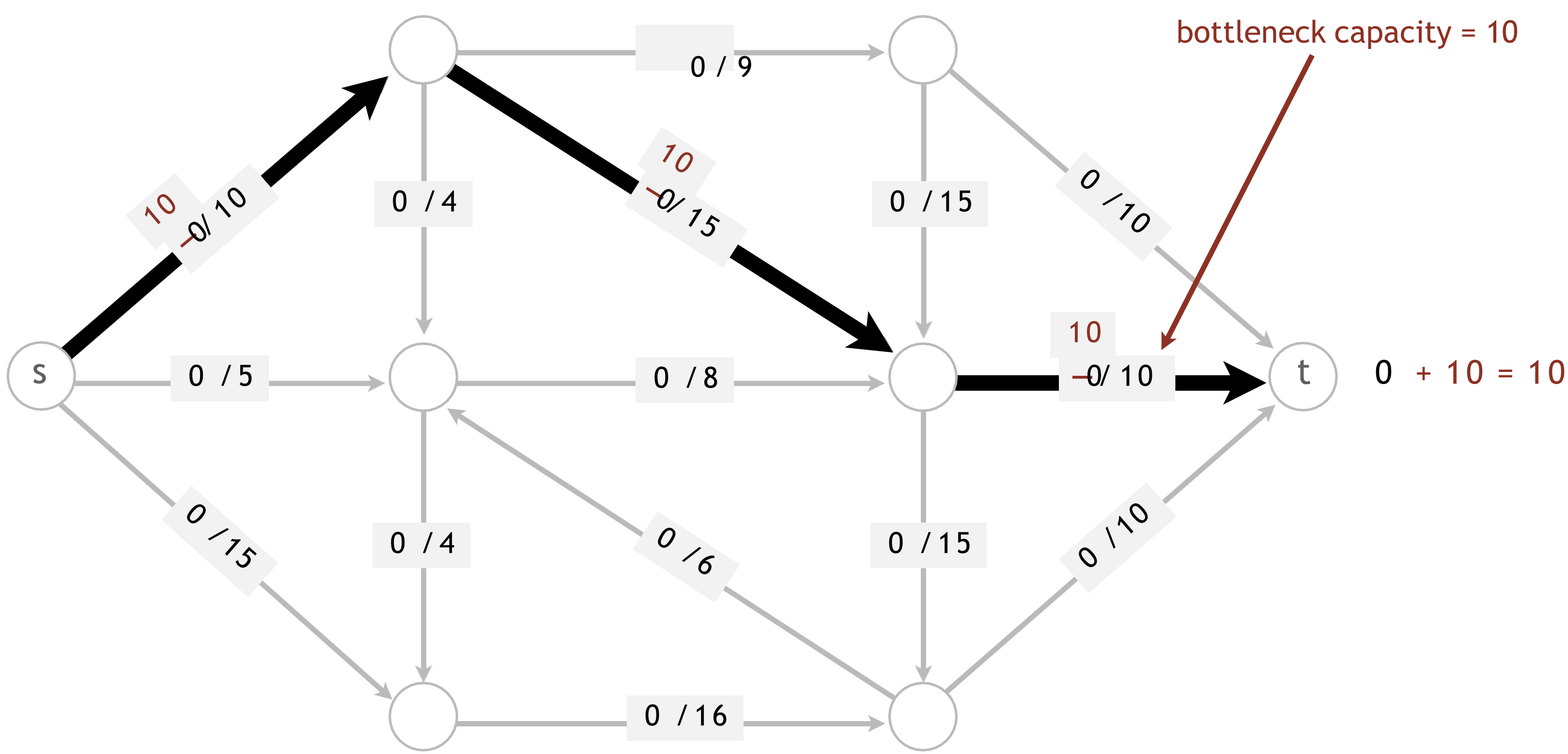


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

**1st augmenting path**

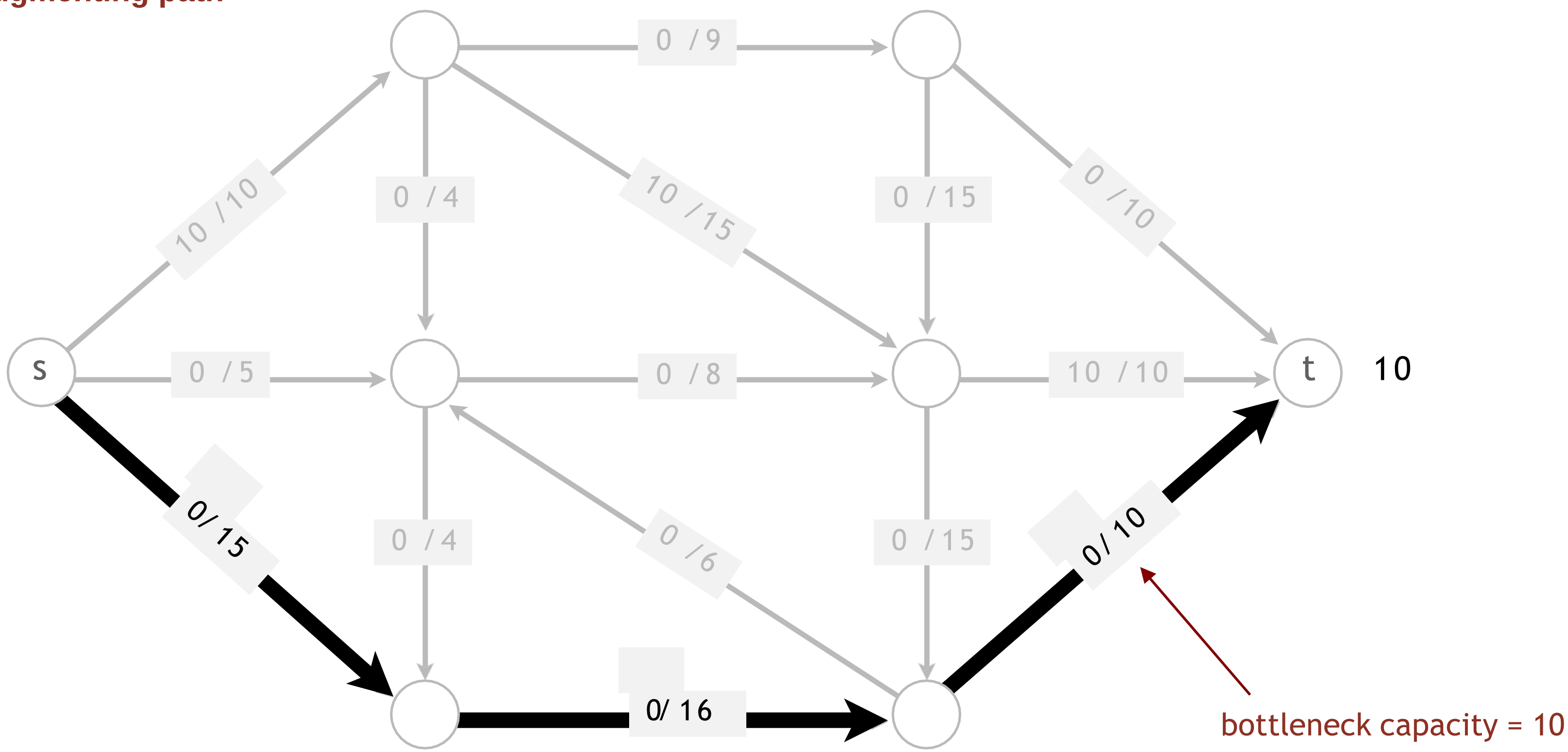


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

**2nd augmenting path**

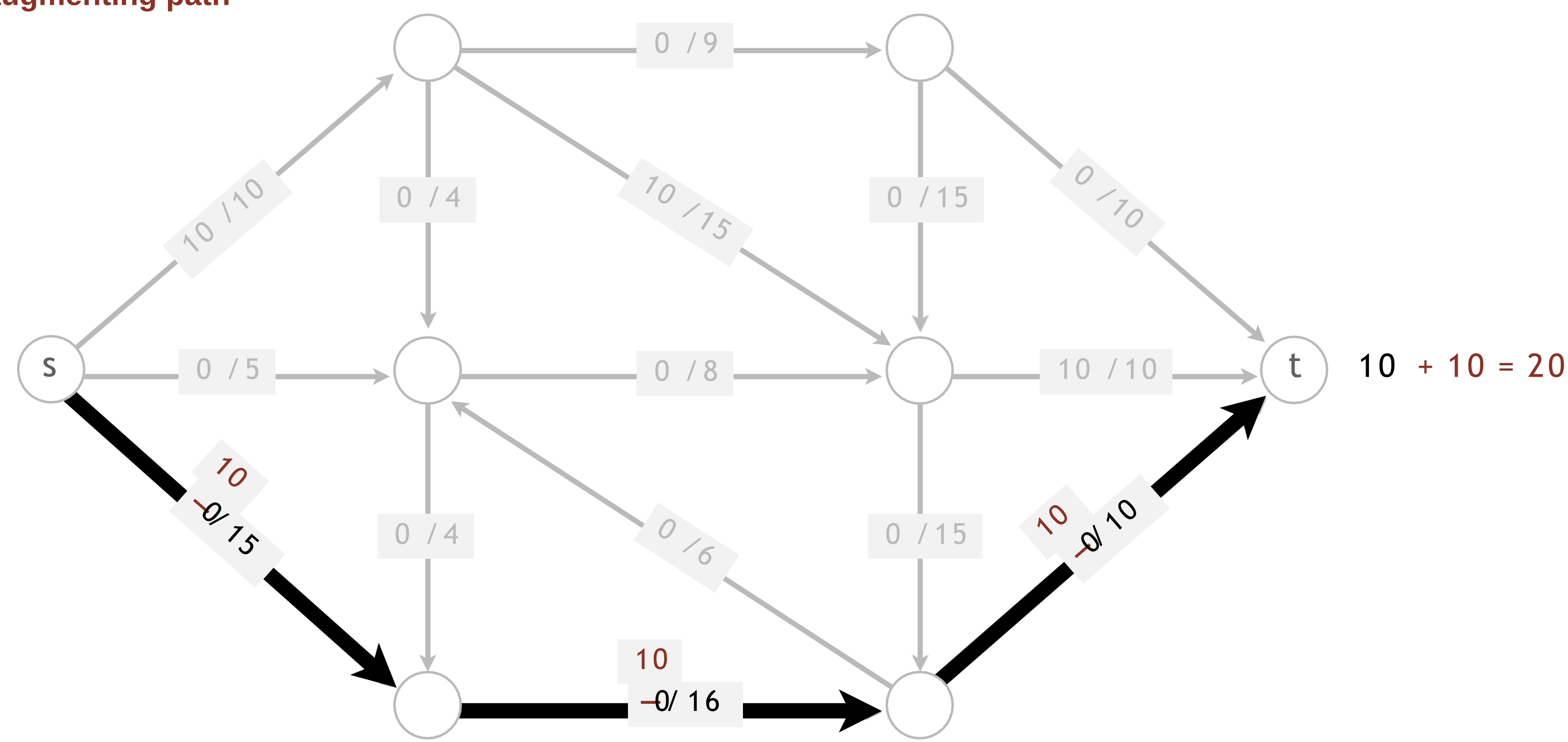


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

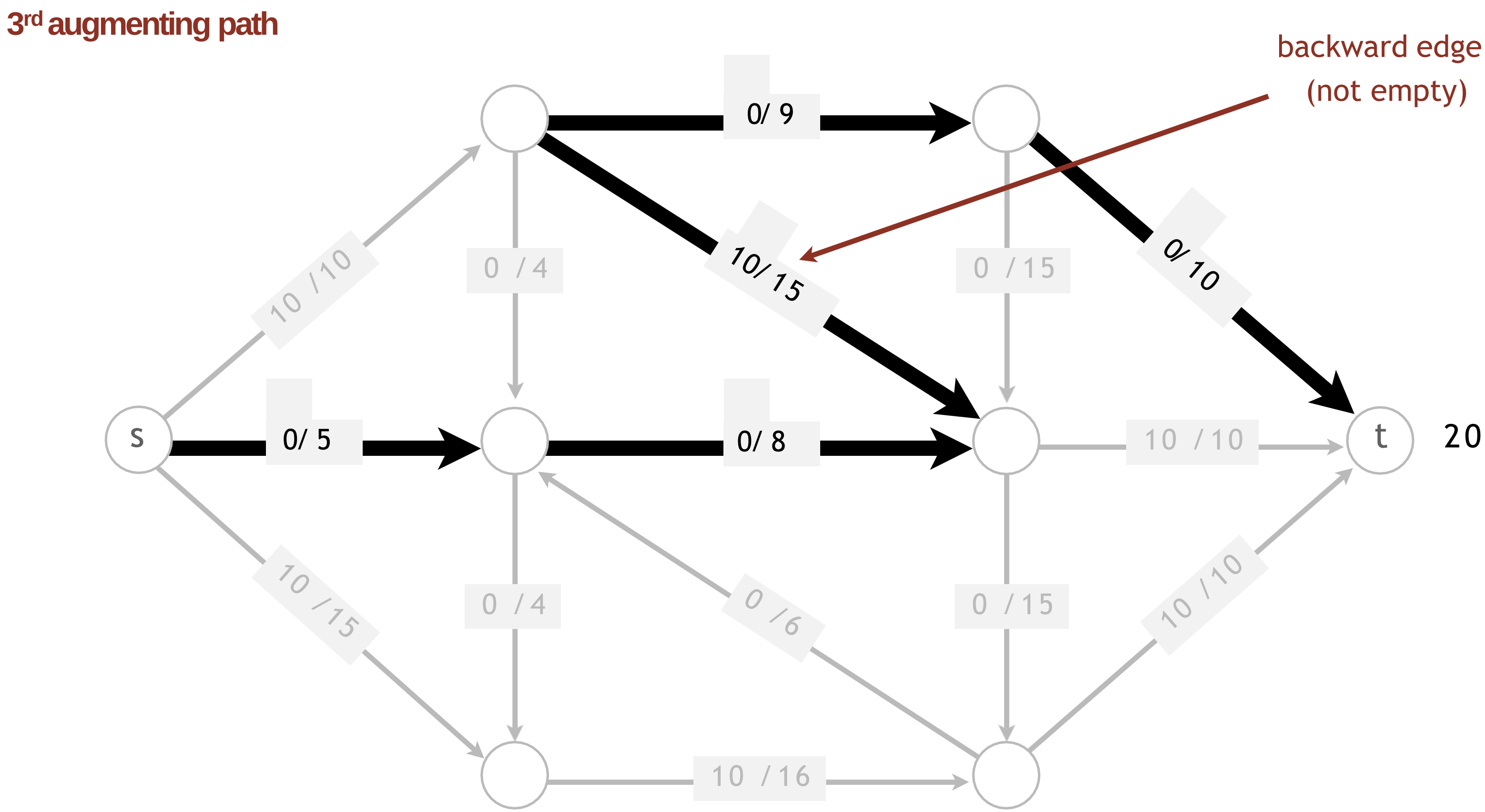
**2nd augmenting path**



# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

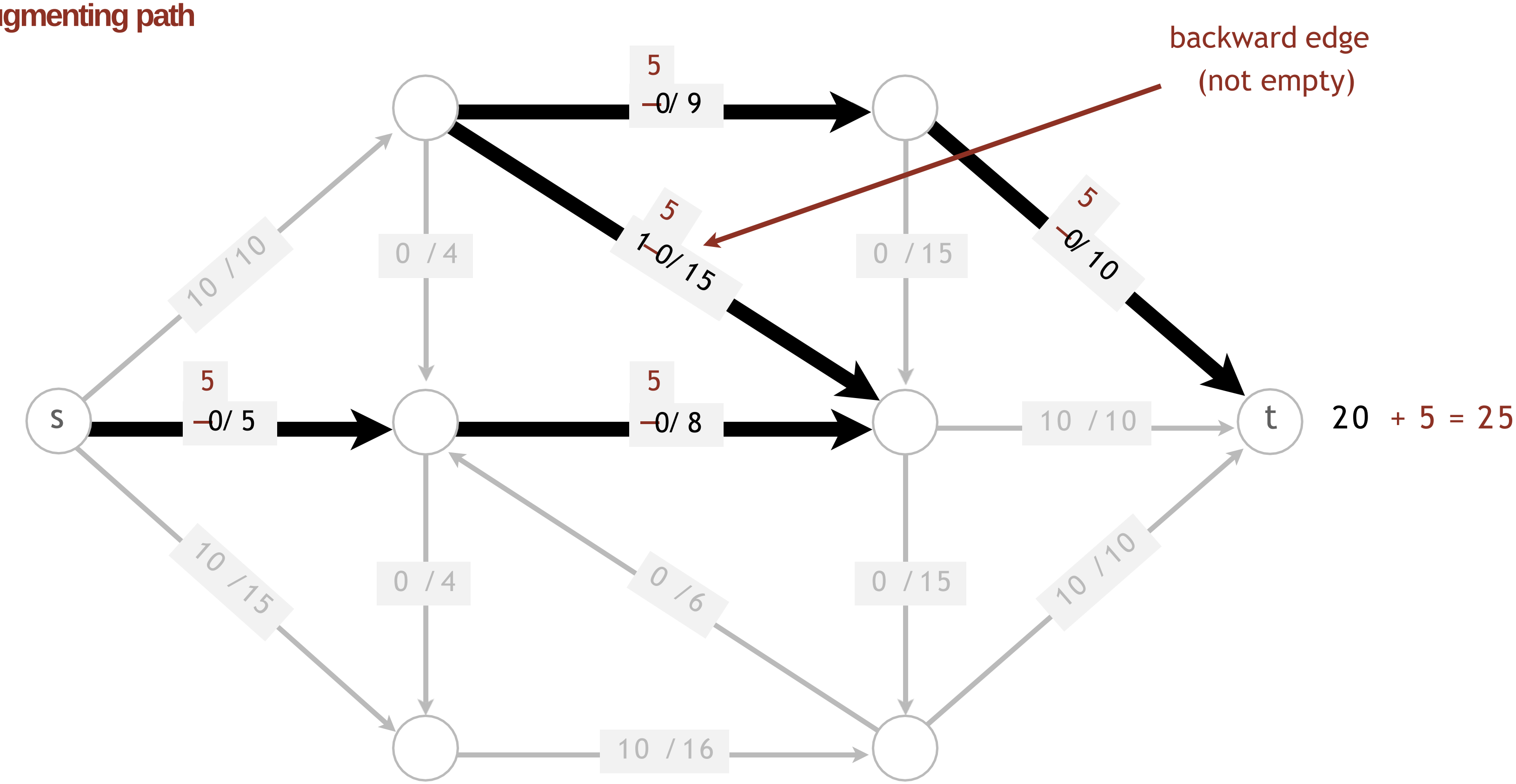


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

3rd augmenting path

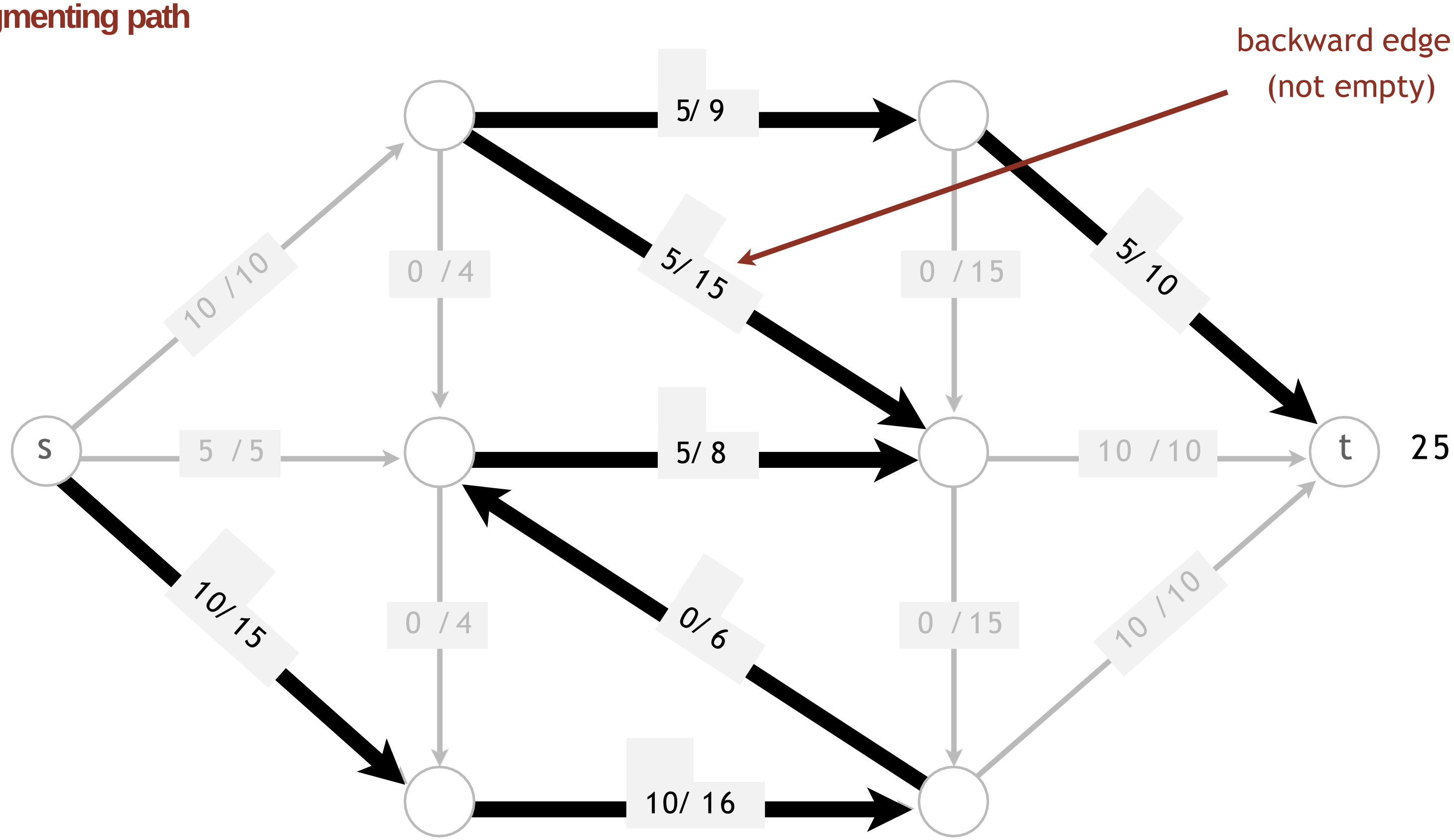


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

4th augmenting path

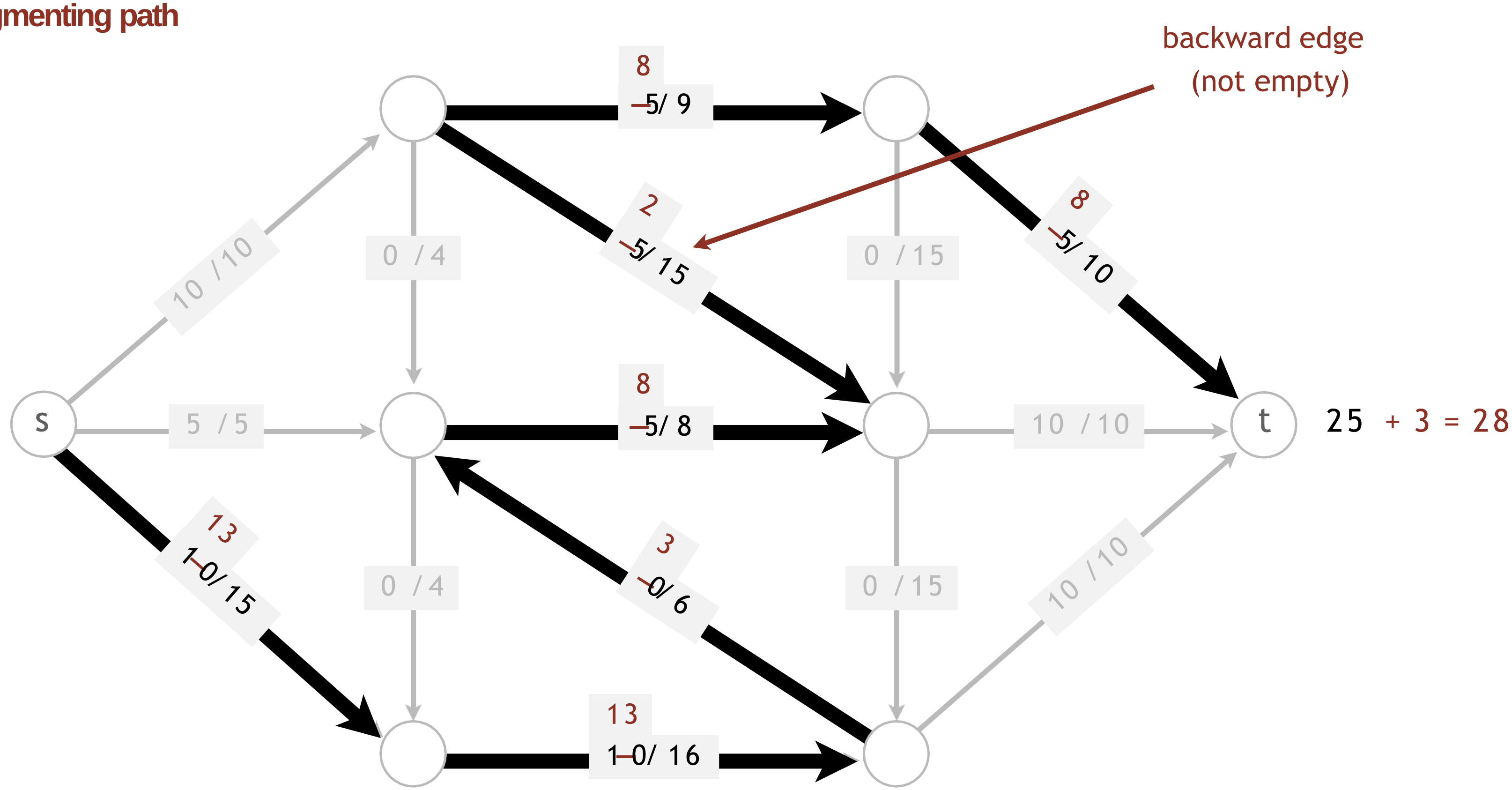


# Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

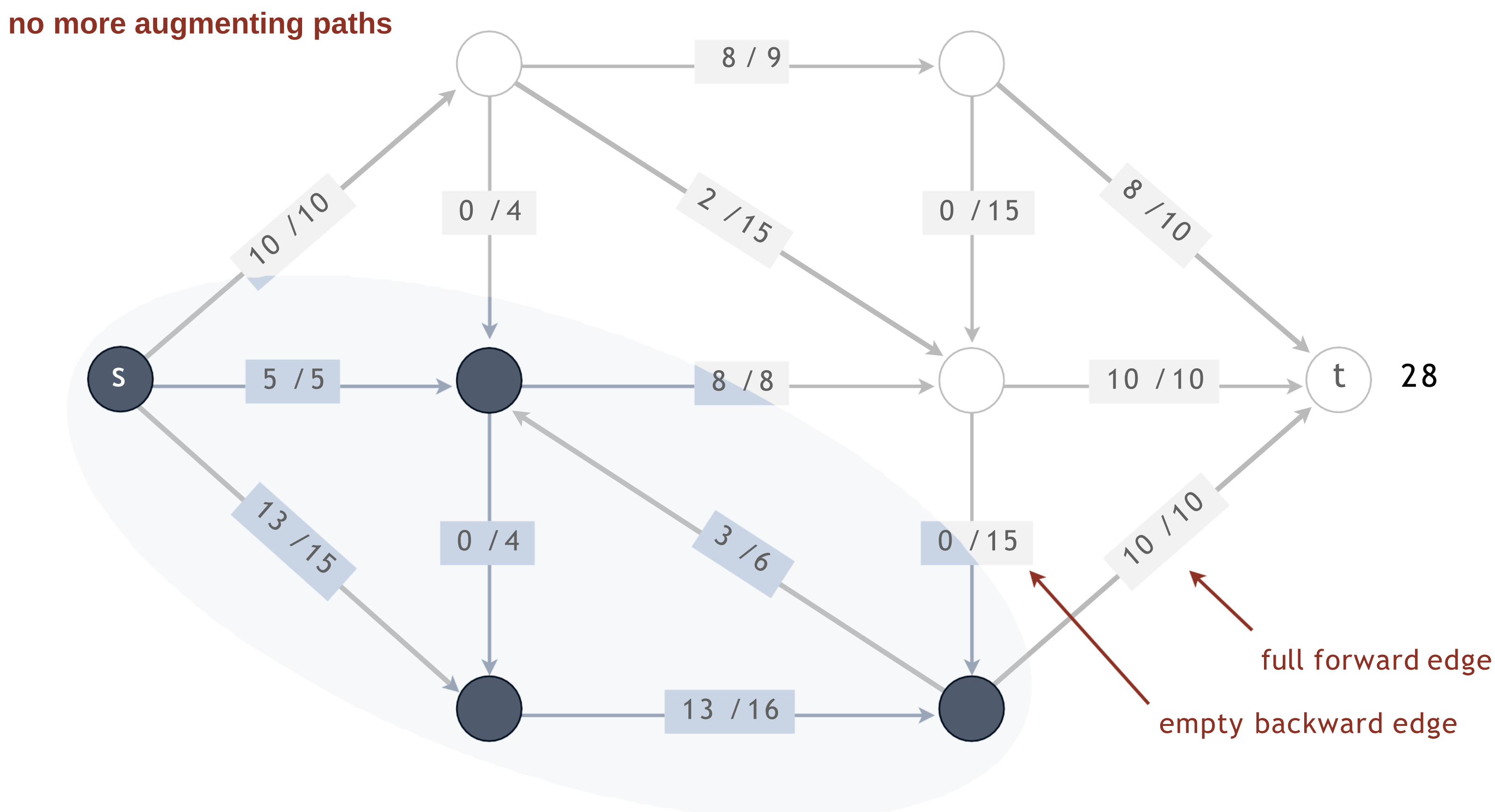
4th augmenting path



# Idea: increase flow along augmenting paths

Termination. All paths from $s$ to $t$ are blocked by either a

- Full forward edge.
- Empty backward edge.



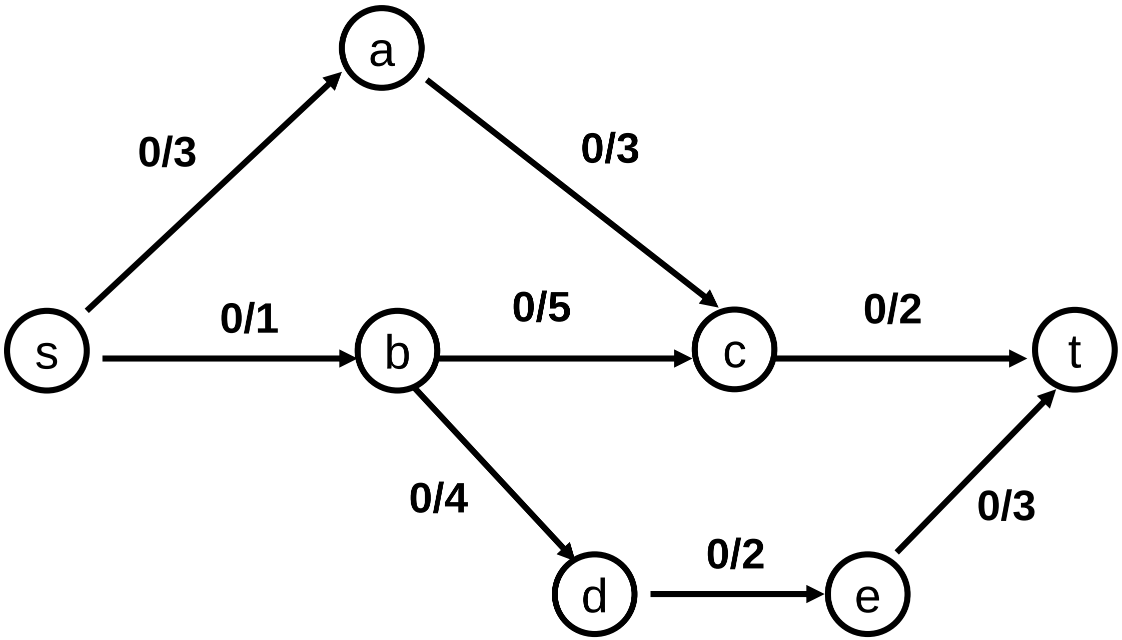
## Complexity of Ford-Fulkerson algorithm

Finding an augmenting path requires a depth-first search of the graph, which takes O(E) time. We have to find a new augmenting path each time the algorithm does another iteration.

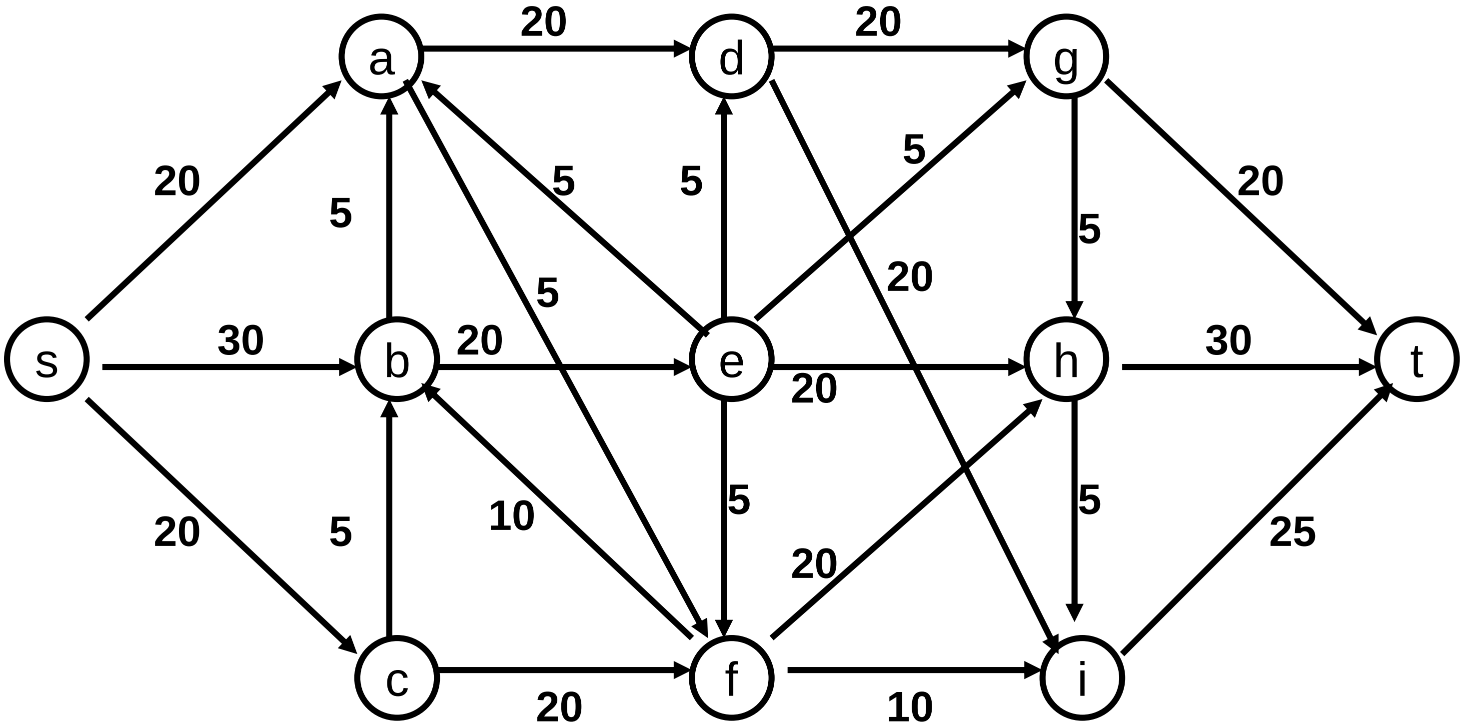
Since we can do at most **f** iterations, and each iteration takes O(E) time, worst case run-time is O(Ef).

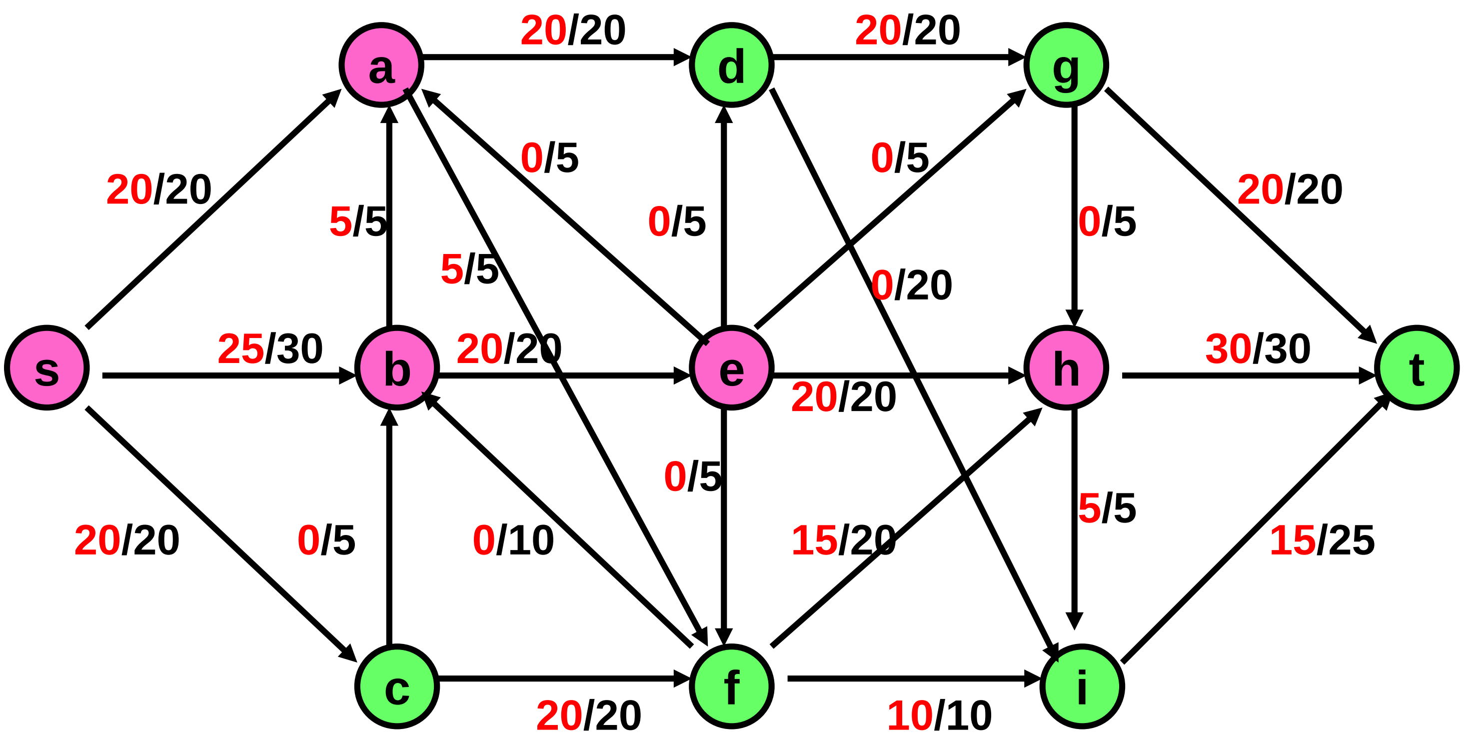
aa

# Find a maximum flow



# Find a maximum flow

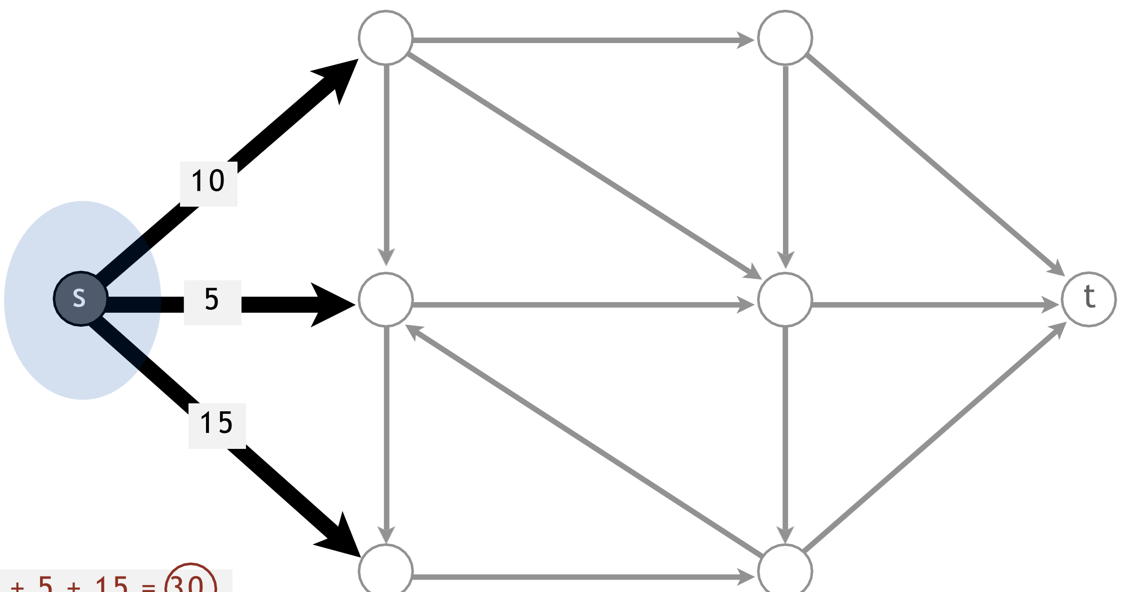



s
a
b
c
d
e
f
g
h
i
t
20/20
25/30
20/20
20/20
5/5
5/5
0/5
20/20
0/5
0/5
0/5
0/10
0/5
20/20
20/20
0/5
0/20
20/20
15/20
10/10
0/5
5/5
20/20
30/30
15/25

# Mincut Problem

Def. A $st$-cut (cut) is a partition of the vertices into two disjoint sets, with $s$ in one set $A$ and $t$ in the other set $B$.

Def. Its capacity is the sum of the capacities of the edges from $A$ to $B$.

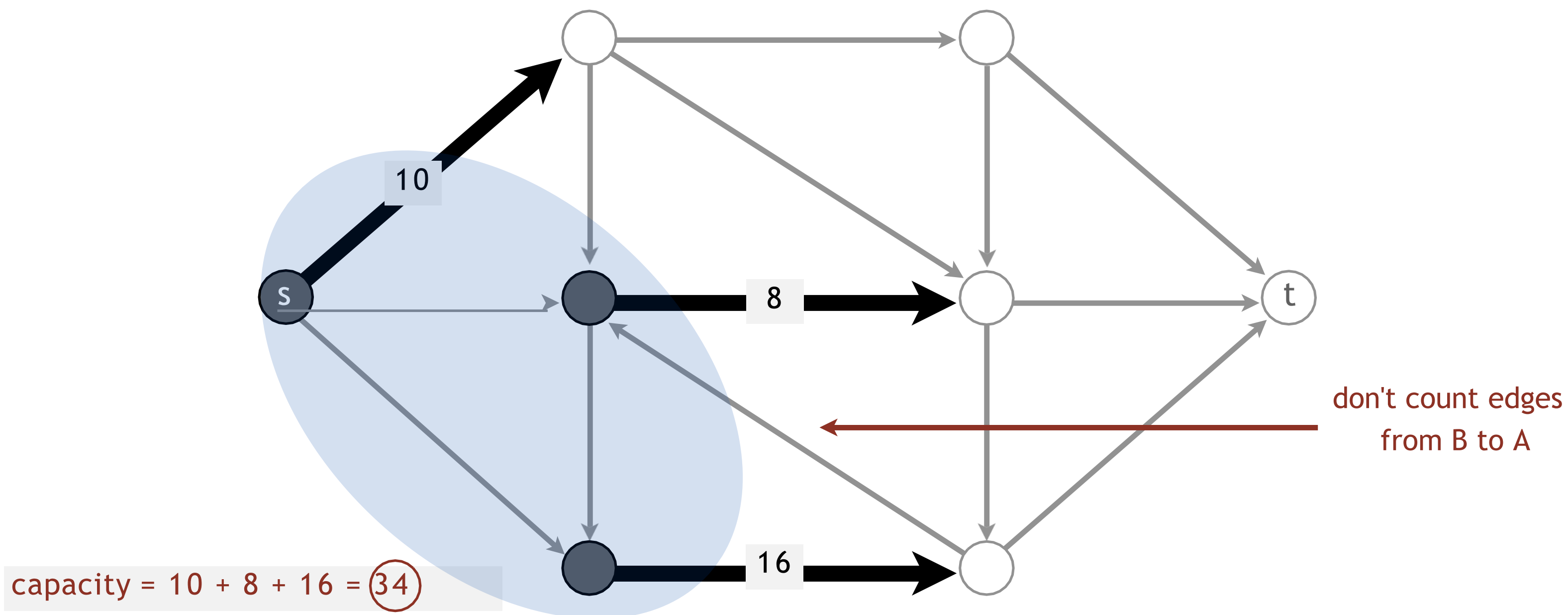



capacity = 10 + 5 + 15 = 30

## Mincut Problem

Def. A *st*-cut (cut) is a partition of the vertices into two disjoint sets, with $s$ in one set $A$ and $t$ in the other set $B$.

Def. Its capacity is the sum of the capacities of the edges from $A$ to $B$.

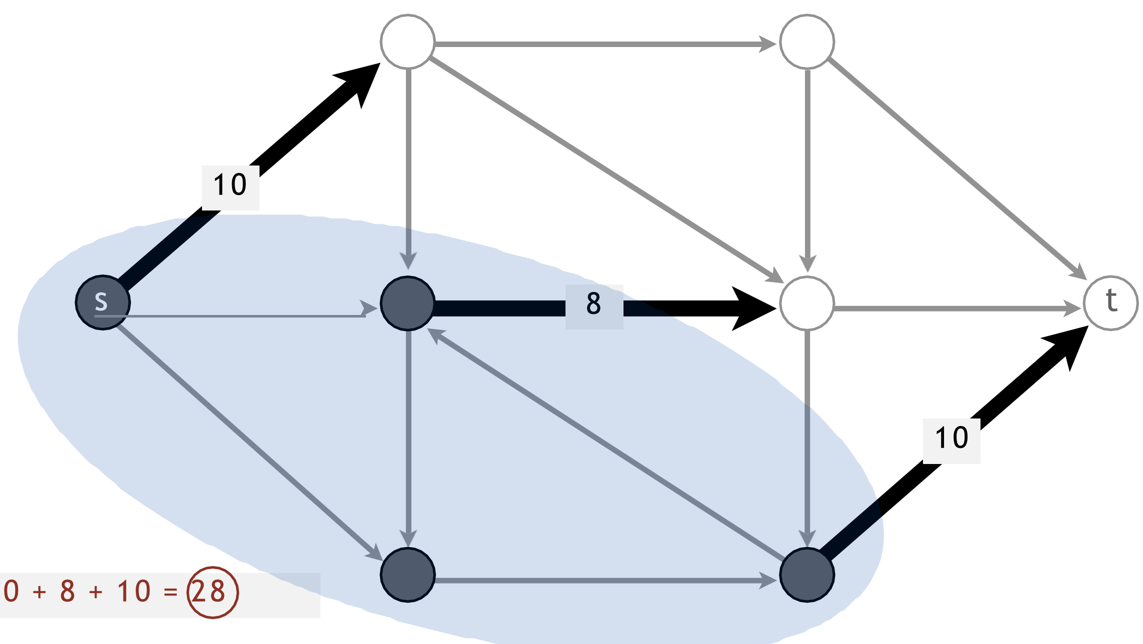


## Mincut Problem

Def. A *st*-cut (cut) is a partition of the vertices into two disjoint sets, with $s$ in one set $A$ and $t$ in the other set $B$.

Def. Its capacity is the sum of the capacities of the edges from $A$ to $B$.

Minimum st-cut (mincut) problem. Find a cut of minimum capacity.




capacity = 10 + 8 + 10 = 28

# Mincut Problem