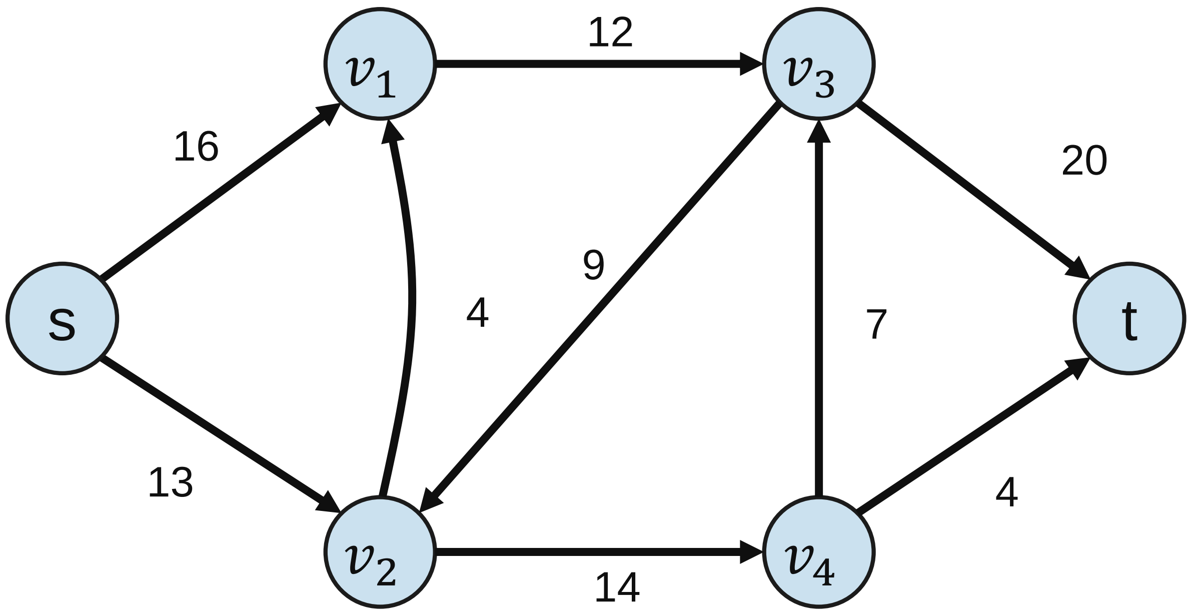


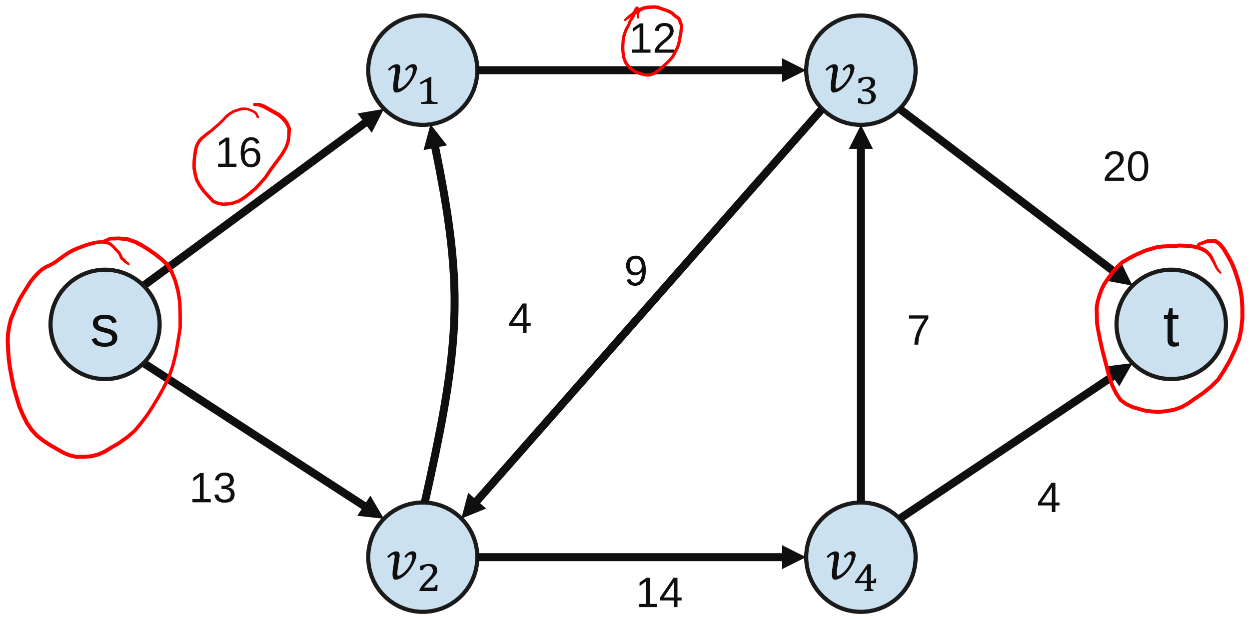
Max Flow

Yannic Maus

Flow network

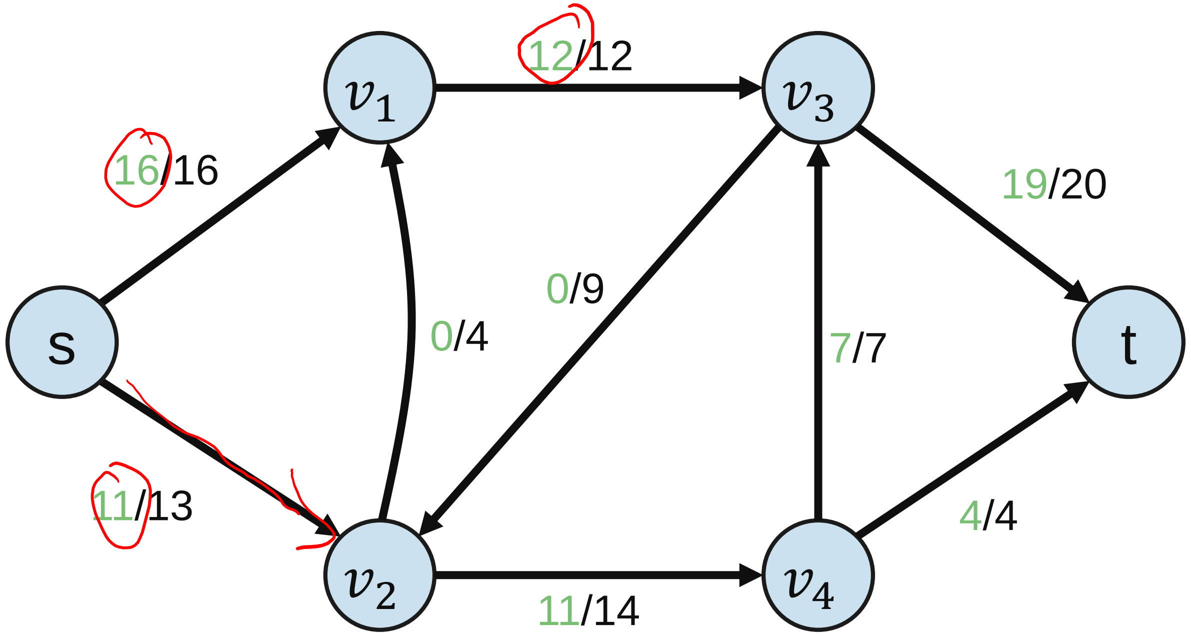
A **flow network** $(\boldsymbol{G}, \boldsymbol{c})$ consists of a directed graph $G = (V, E)$ together with

- a capacity function $c: E \to \mathbb{N}_{\geq 0}$ **(set to $\boldsymbol{c(u,v) = 0}$ for $\boldsymbol{(u,v) \notin E}$)**
- a source s, a sink t: in-degree$(s) = 0$, out-degree$(t) = 0$



A **flow network** $(\boldsymbol{G}, \boldsymbol{c})$ consists of a directed graph $G = (V, E)$ together with

- a capacity function $c: E \to \mathbb{N}_{\geq 0}$ **(set to $\boldsymbol{c}(\boldsymbol{u}, \boldsymbol{v}) = 0$ for $(\boldsymbol{u}, \boldsymbol{v}) \notin \boldsymbol{E}$)**
- a source s, a sink t: in-degree$(s) = 0$, out-degree$(t) = 0$

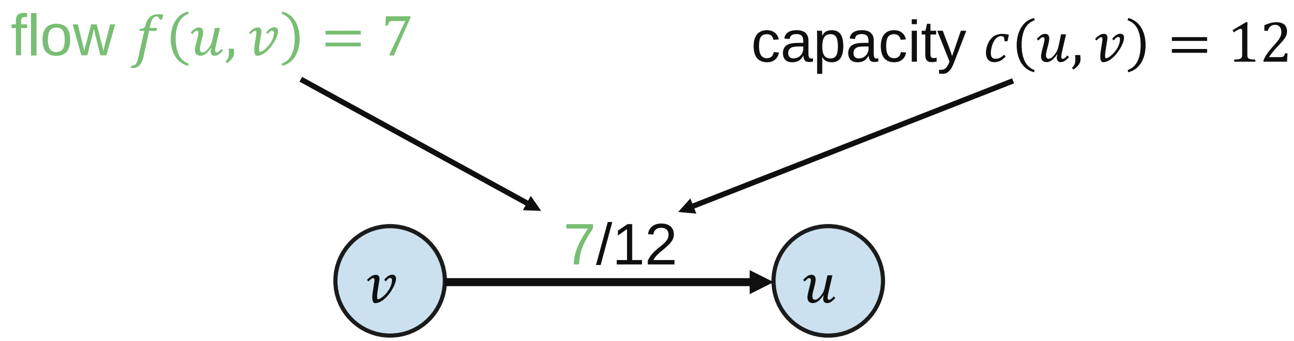


A flow in a flow network (G, c) is a function $f: V \times V \to \mathbb{R}$ that satisfies

- **capacity constraints,**
- **conservation of flow,**
- **skew symmetry.**

Capacity constraint

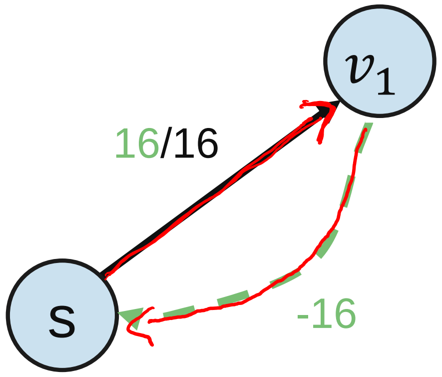
Capacity constraint: $f(u,v) \leq c(u,v)\ \forall (u,v) \in V \times V$



Never exceed the capacity of an edge

(technicality: we might have $c(u,v) = 0$, and $f(u,v) < 0$)

Skew symmetry: $f(u,v) = -f(v,u)$

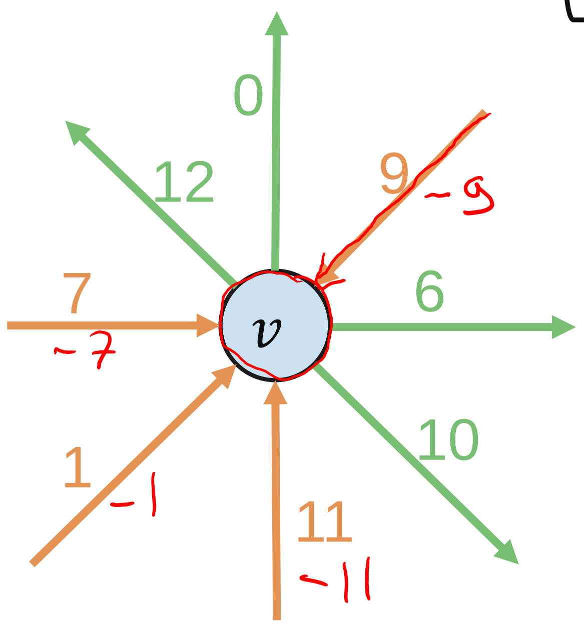


We only note down positive flows.
The other direction of flow is "implicit"

The net flow from u to v must be the opposite of the net flow from v to u.

Conservation of flow

Conservation of flow: $\forall v \in V \setminus \{s,t\}$: $\underbrace{f(v,V) = \sum_{u\in V} f(v,u) = 0}$



$f(v,V) = -f_{in}(v) + f_{out}(v) = 0$

$f_{in}(v)$ =9+7+11+1=28

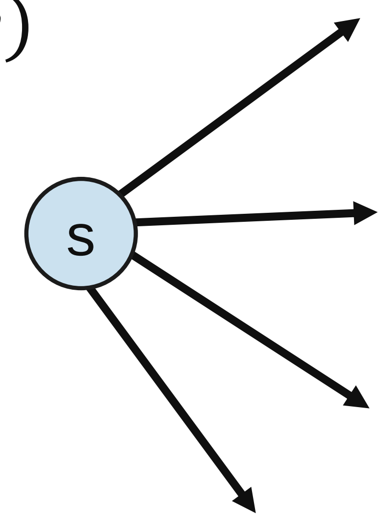
$f_{out}(v)$ =6+10+12=28

The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow

Value of a flow

The value $|f|$ of a flow f is defined as $|f| = \sum_{v \in V \setminus \{s\}} f(s, v)$

"the flow leaving the source"

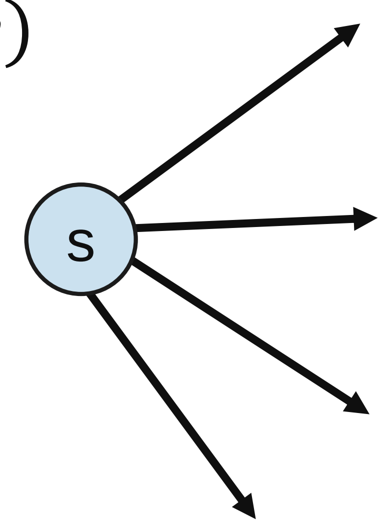


Goal: Given a flow network, find a flow with maximum value

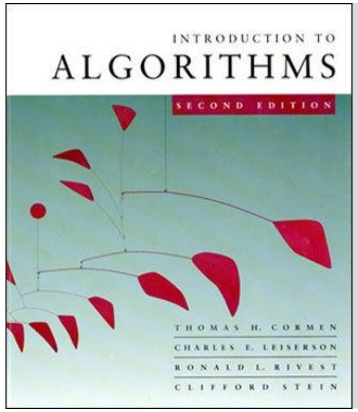
Value of a flow

The value $|f|$ of a flow f is defined as $|f| = \sum_{v \in V \setminus \{s\}} f(s, v)$

“the flow leaving the source”



Goal: Given a flow network, find a flow with maximum value



Warning! Not all literature have the exact same definition of a flow, e.g., CLRS requires $f(u, v) \geq 0$ for all $u, v \in V$, while we explicitly allow and make use of $\underline{f(u, v) < 0}$ (recall skew symmetry).

Both definitions lead to a solid theory and arguments are similar, but differ a bit in the details.

Agenda

- Ford-Fulkerson-Method
  - Correctness & Termination
  - Bad Examples
- Edmonds-Karp-Algorithm
  - Runtime analysis
- Outlook: State of the art …
- Application: Maximum Cardinality Bipartite Matchings

Ford-Fulkerson-Method

Ford-Fulkerson Method

Input: Given a flow network (G, c) with source node s, and sink node t
Output: Compute a flow f from s to t of maximum value

Ford-Fulkerson Method

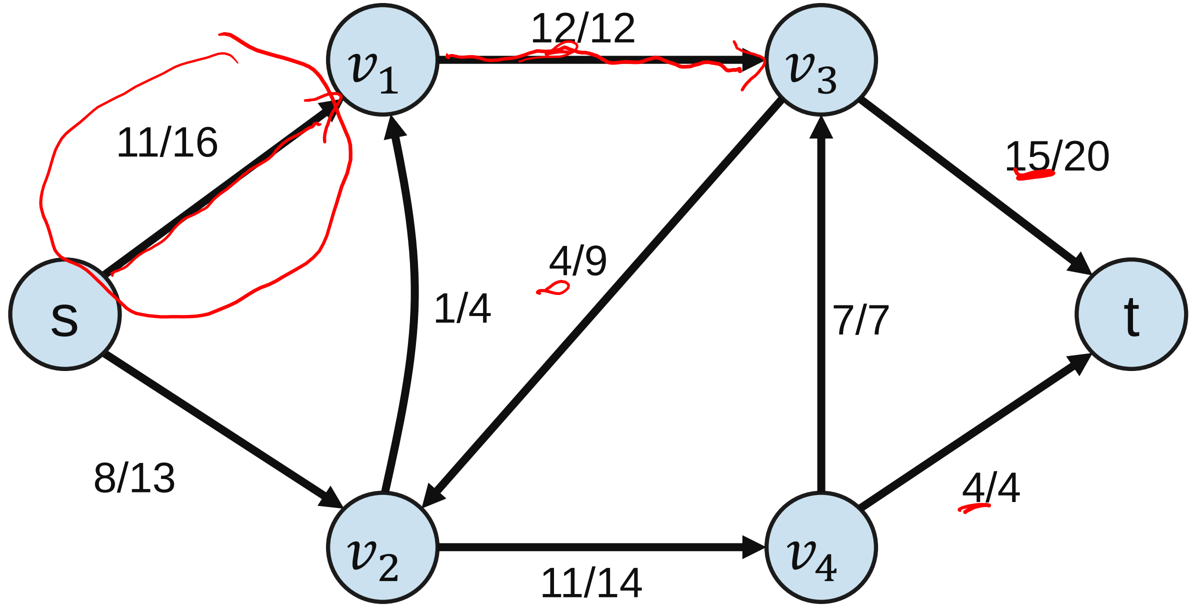
Input: Given a flow network (G, c) with source node s, and sink node t
Output: Compute a flow f from s to t of maximum value

```
Initialize flow f with 0
while there exists an augmenting path p in G_f do
    augment f along p (by f_p)
return f
```

Residual Network

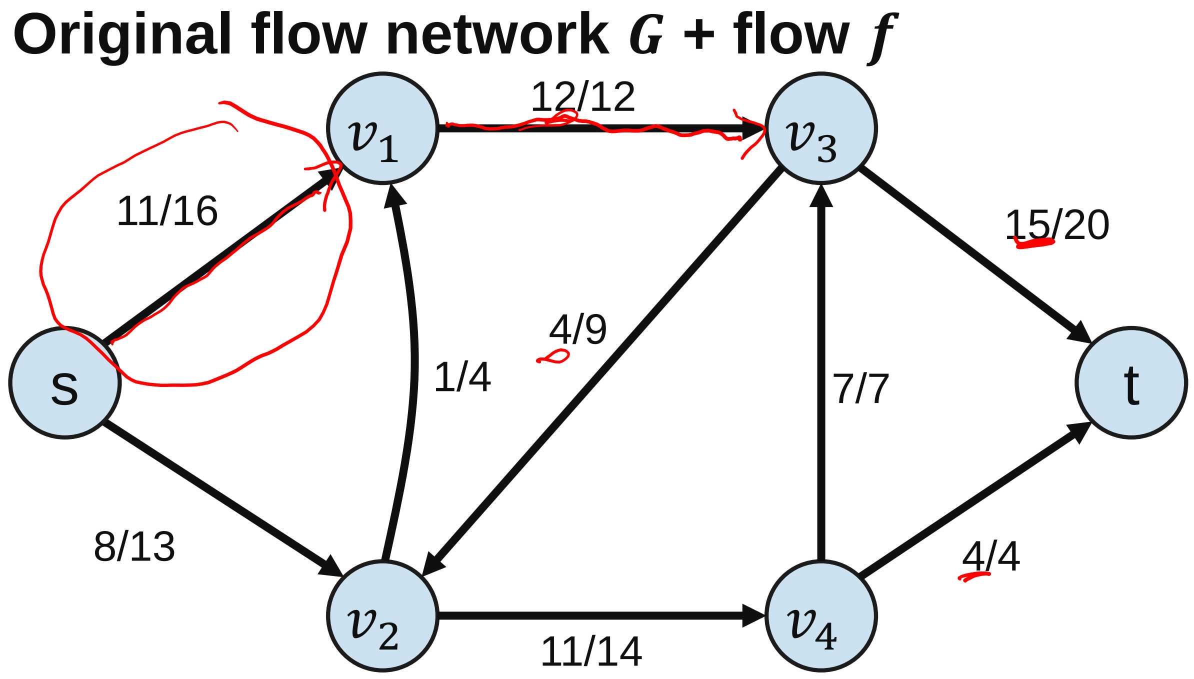
Given a flow network (G, c) and a flow f: for **all pairs** $(u, v) \in V \times V$ define the **residual capacity** of f $\boldsymbol{c_f(u, v) = c(u, v) - f(u, v)}$

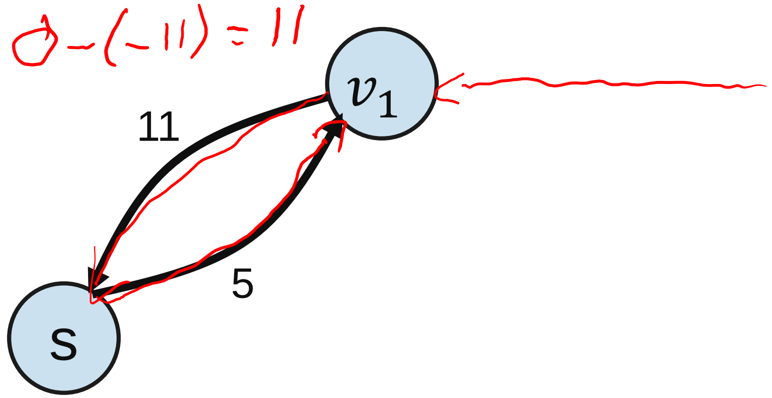
Original flow network G + flow f



$0-(-11)=11$

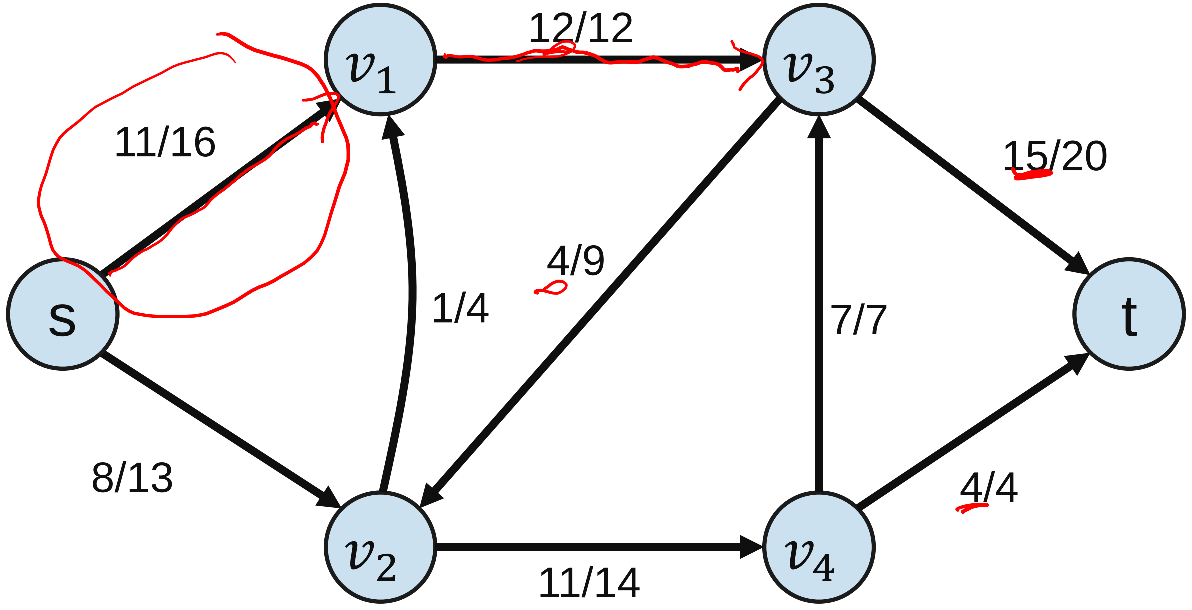
Given a flow network (G, c) and a flow f: for **all pairs** $(u, v) \in V \times V$ define the **residual capacity** of f $\boldsymbol{c_f(u,v) = c(u,v) - f(u,v)}$





Given a flow network (G, c) and a flow f: for **all pairs** $(u, v) \in V \times V$ define the **residual capacity** of f $\boldsymbol{c_f(u,v) = c(u,v) - f(u,v)}$

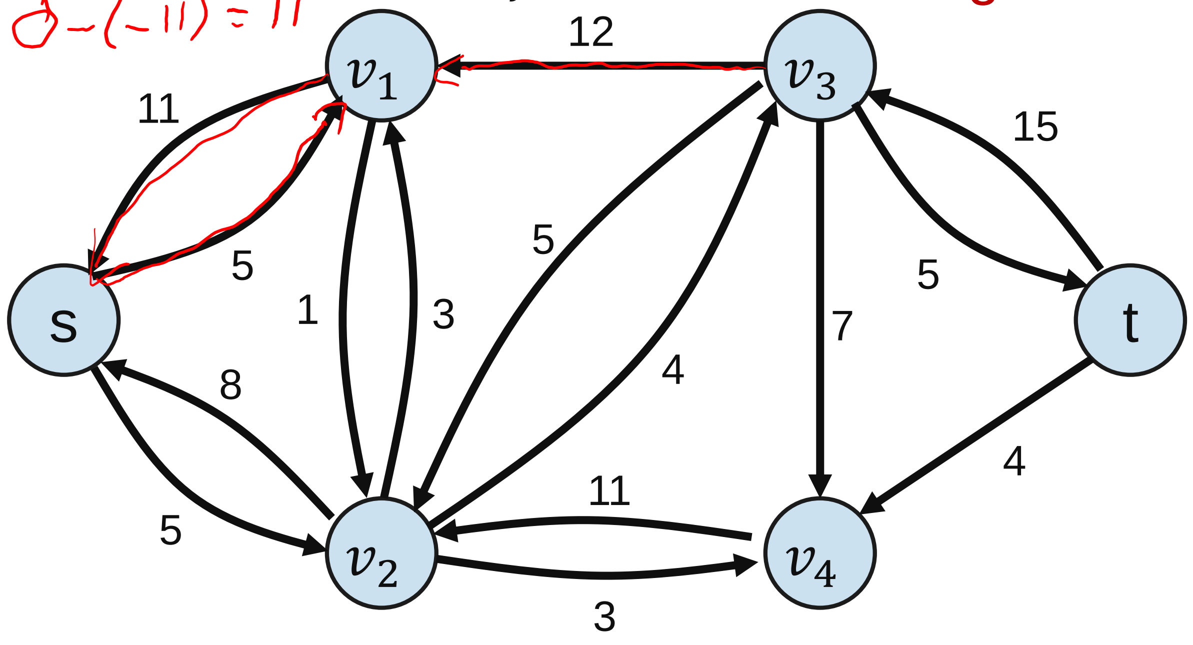
Original flow network G + flow f



Residual network G_f

backward edges!

$0-(-11)=11$



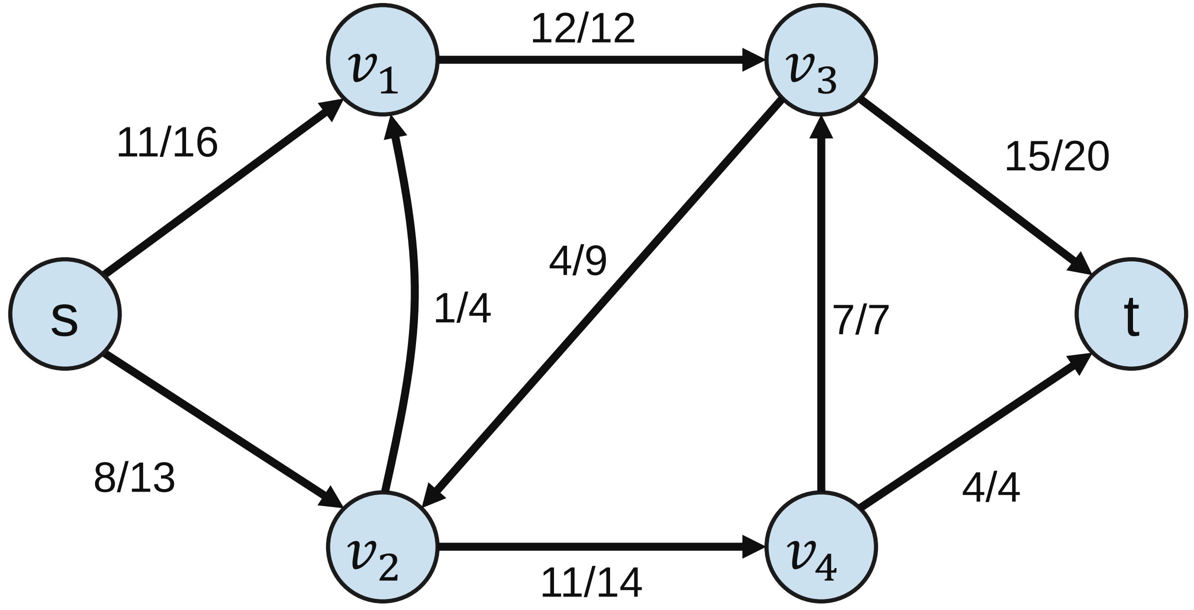
Given a flow network (G, c) and a flow f: for **all pairs** $(u, v) \in V \times V$ define the **residual capacity** of f $\boldsymbol{c_f(u,v) = c(u,v) - f(u,v)}$

The **residual network** $G_f(V, E_f)$ is given via the edge set

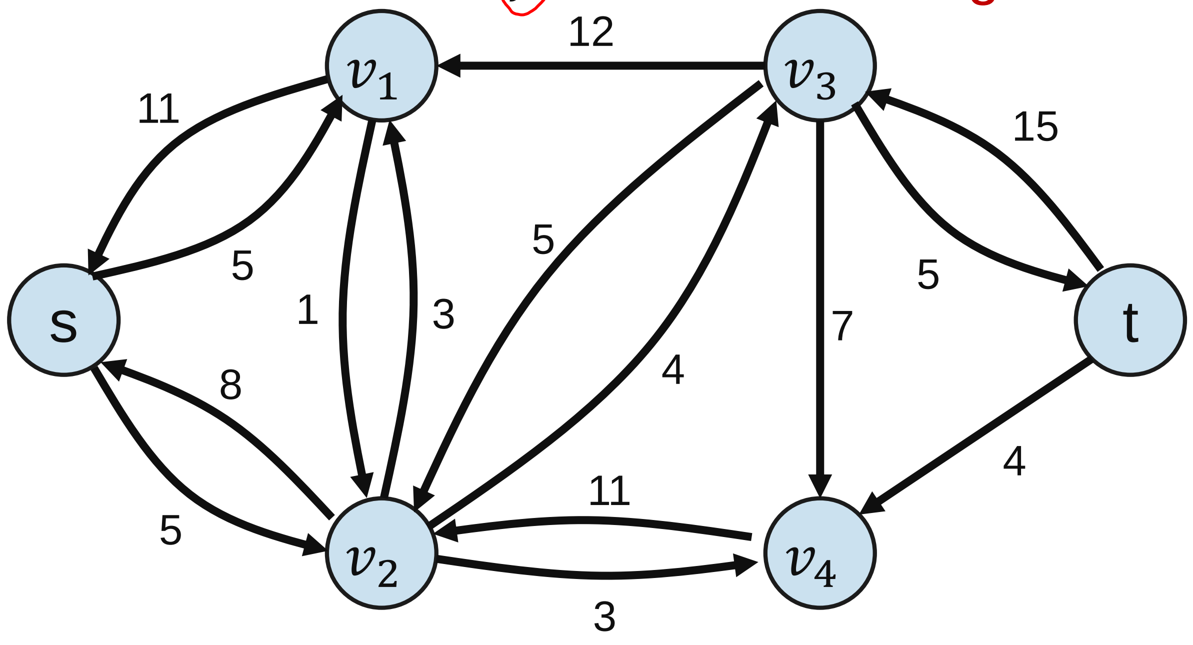
$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$$

G_f is a flow network with capacities c_f.

Original flow network G + flow f



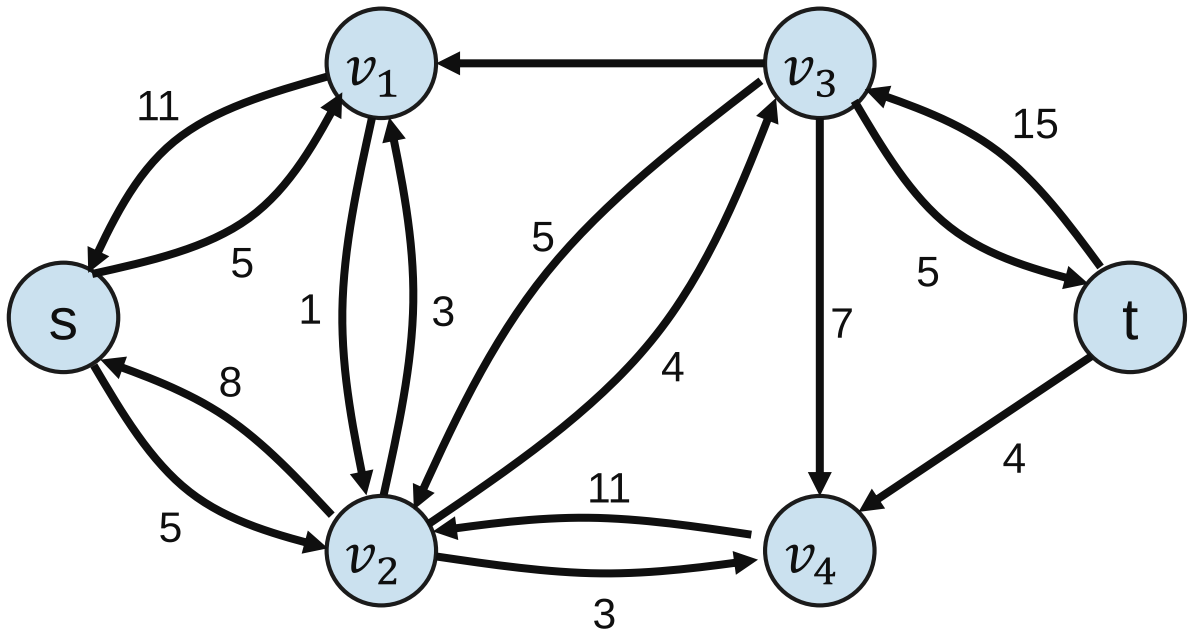
Residual network G_f backward edges!



Augmenting path

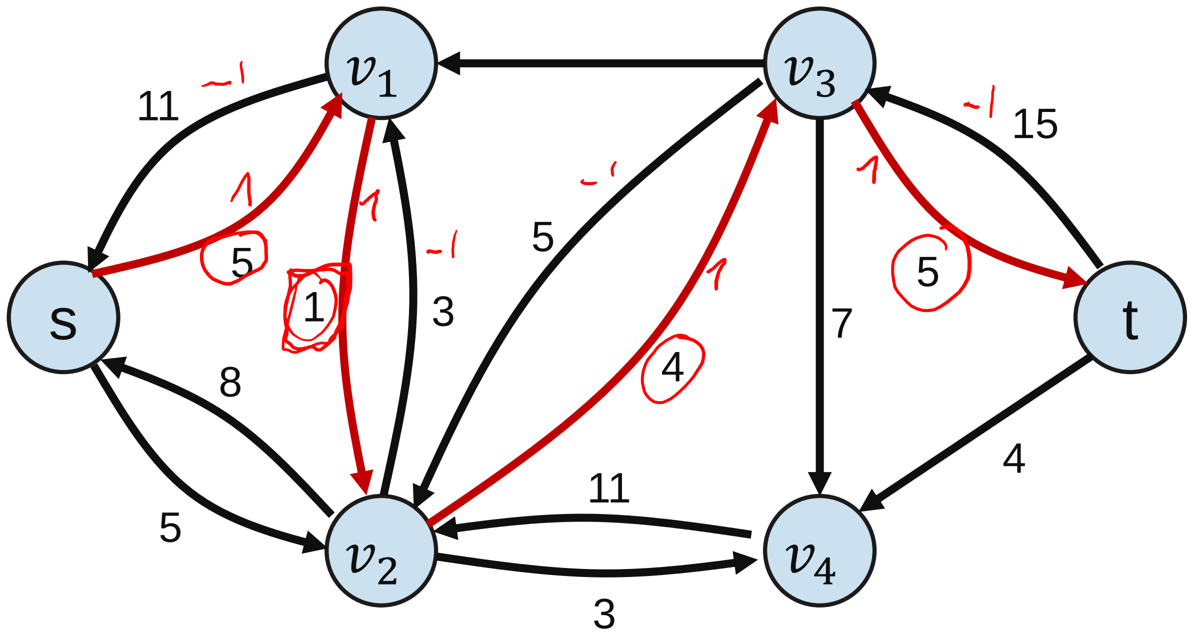
An s-t-path p in G_f is an augmenting path.

$$c_f(p) = \min\{\, c_f(u,v) \,|\, (u,v) \in p \,\}$$ is the **residual capacity** of p.



An s-t-path p in G_f is an augmenting path.

$c_f(p) = \min\{\, c_f(u,v) \mid (u,v) \in p \,\}$ is the **residual capacity** of p.



Residual capacity $c_f(p)$ of this path is 1

Ford-Fulkerson Method

Input: Given a flow network (G, c) with source node s, and sink node t
Output: Compute a flow f from s to t of maximum value

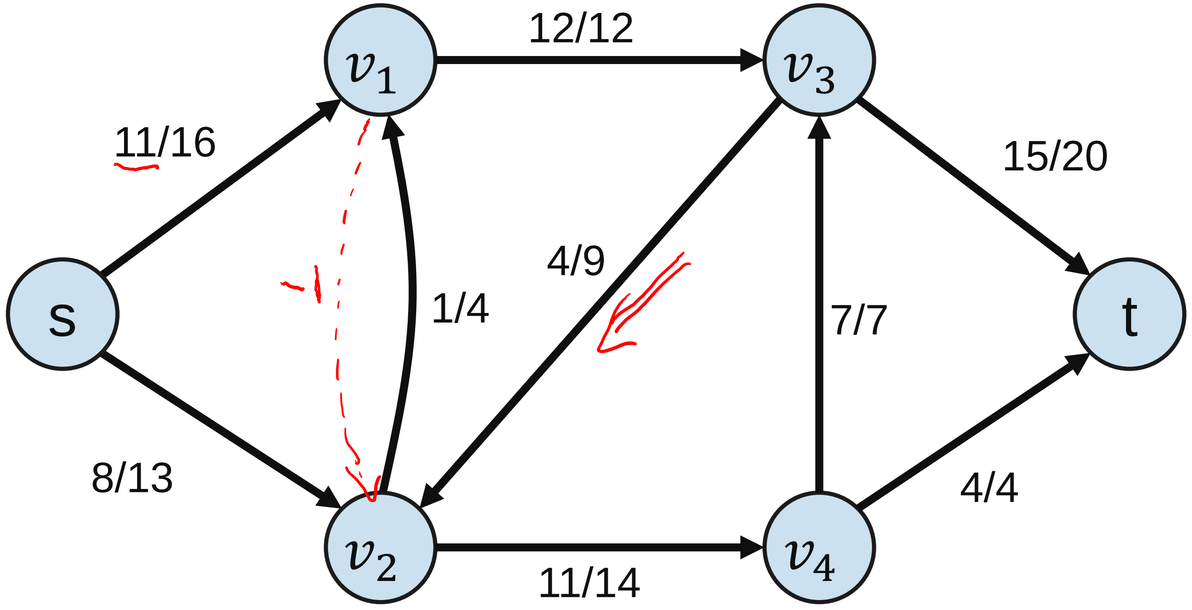
```
Initialize flow f with 0
while there exists an augmenting path p in G_f do
    augment f along p (by f_p)
return f
```

$$f_p(u,v) = \begin{cases} c_p(u,v) & \text{if } (u,v) \in p \\ -c_p(u,v) & \text{if } (v,u) \in p \\ 0 & \text{otherwise} \end{cases}$$

$\Rightarrow$ f_p is a flow in G_f

augment f along p: $f + f_p$

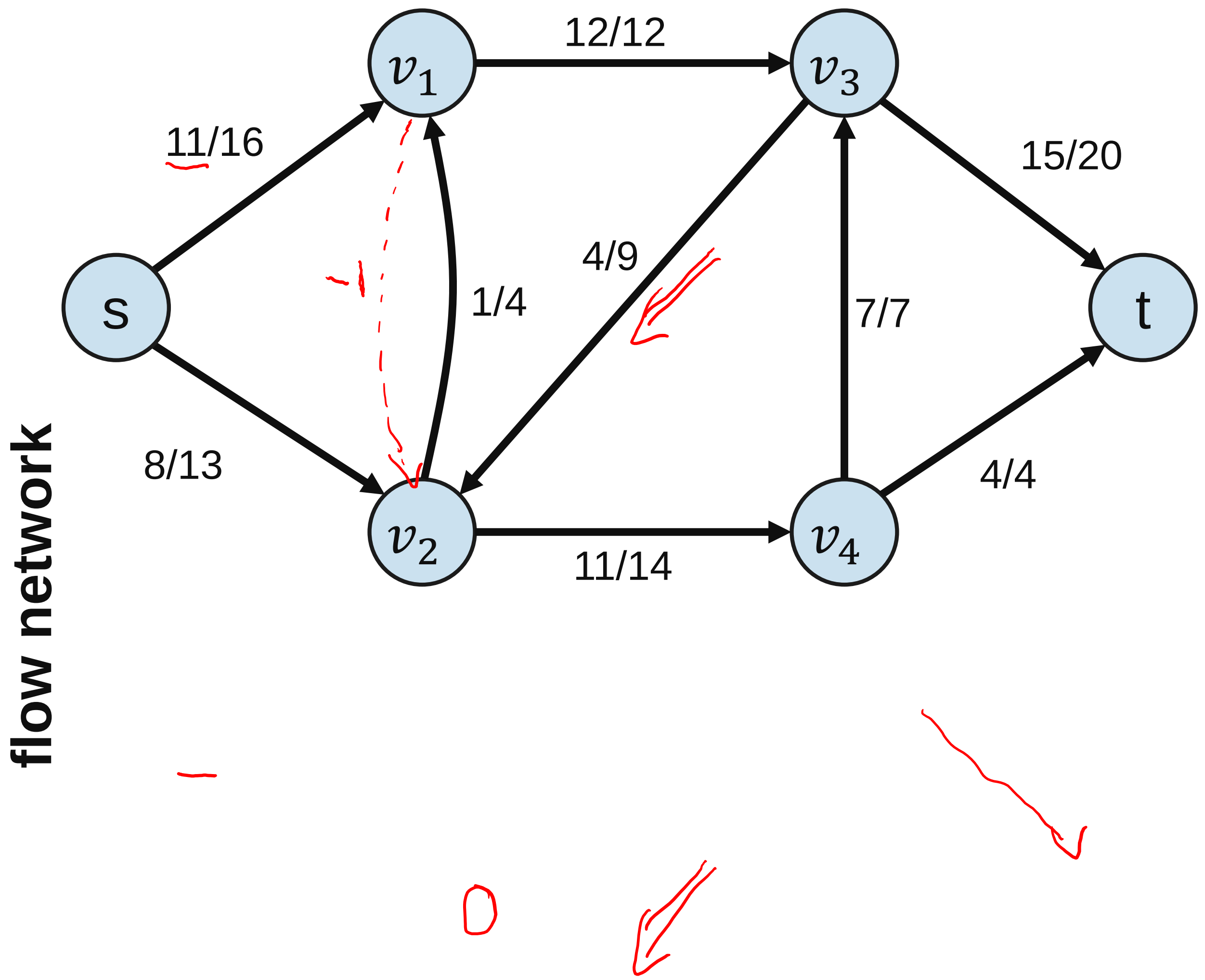
Example: Ford-Fulkerson-Method

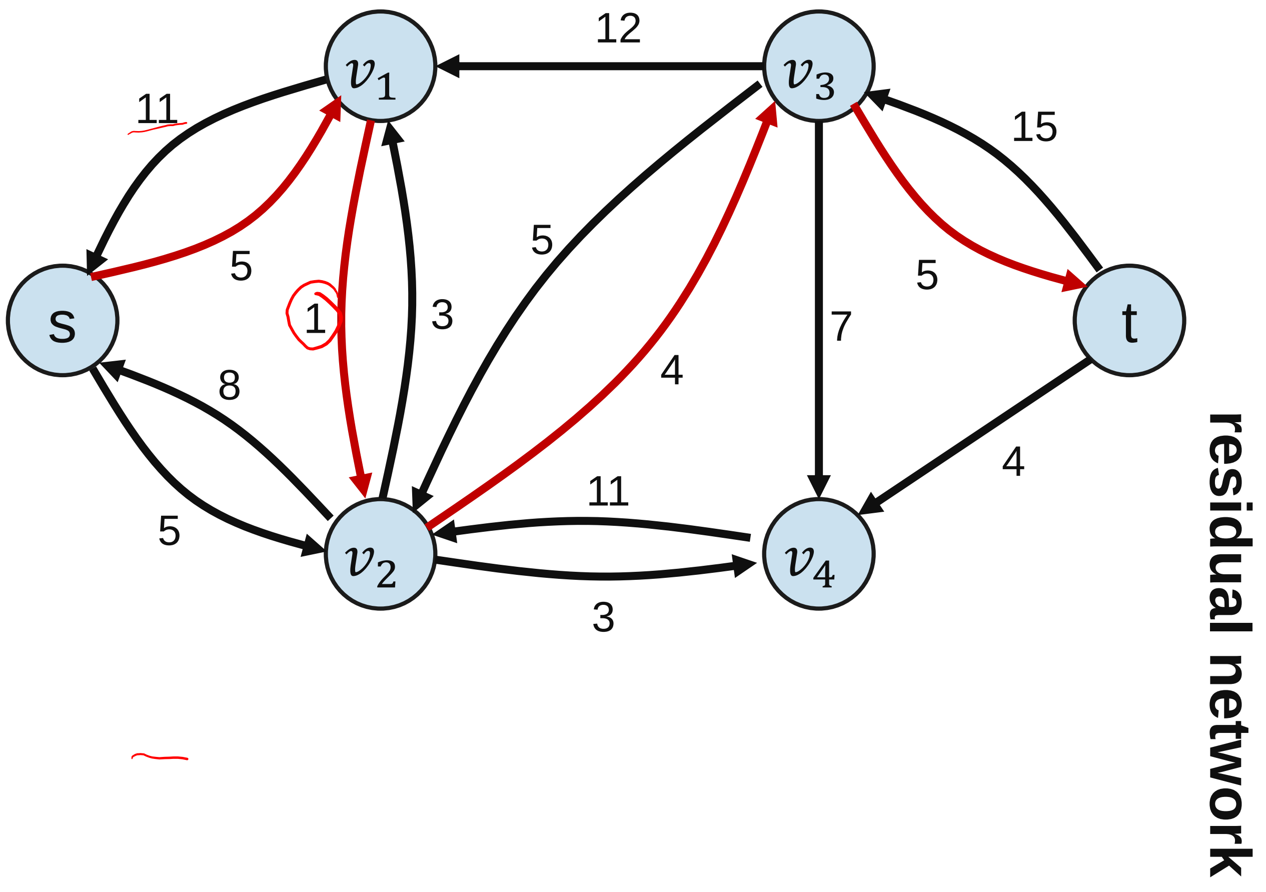


flow network

residual network

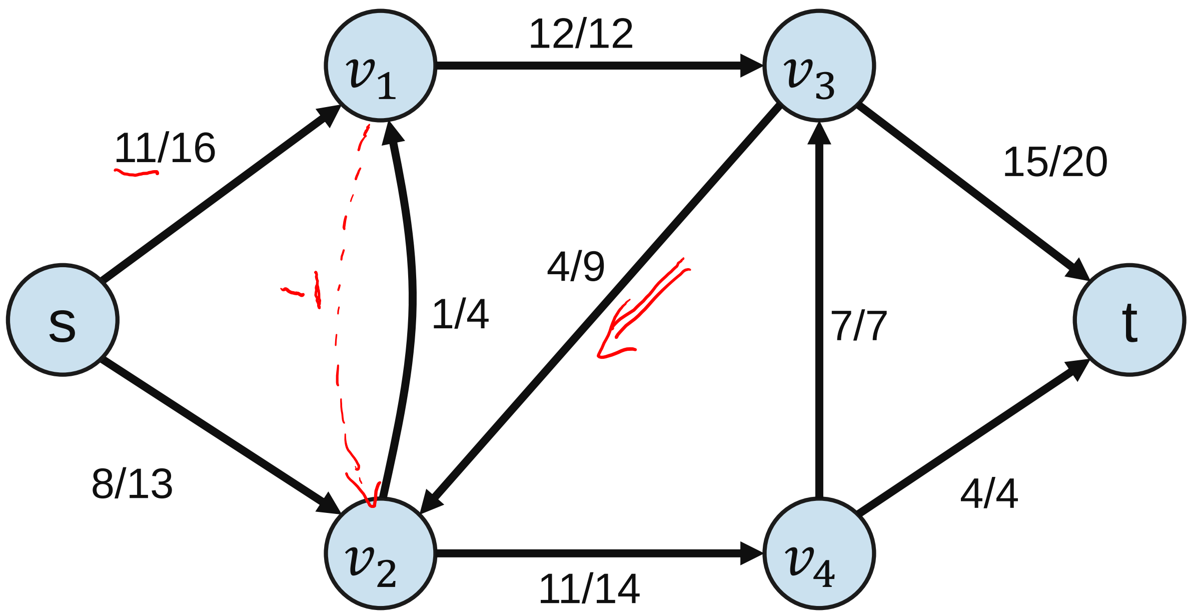
Example: Ford-Fulkerson-Method

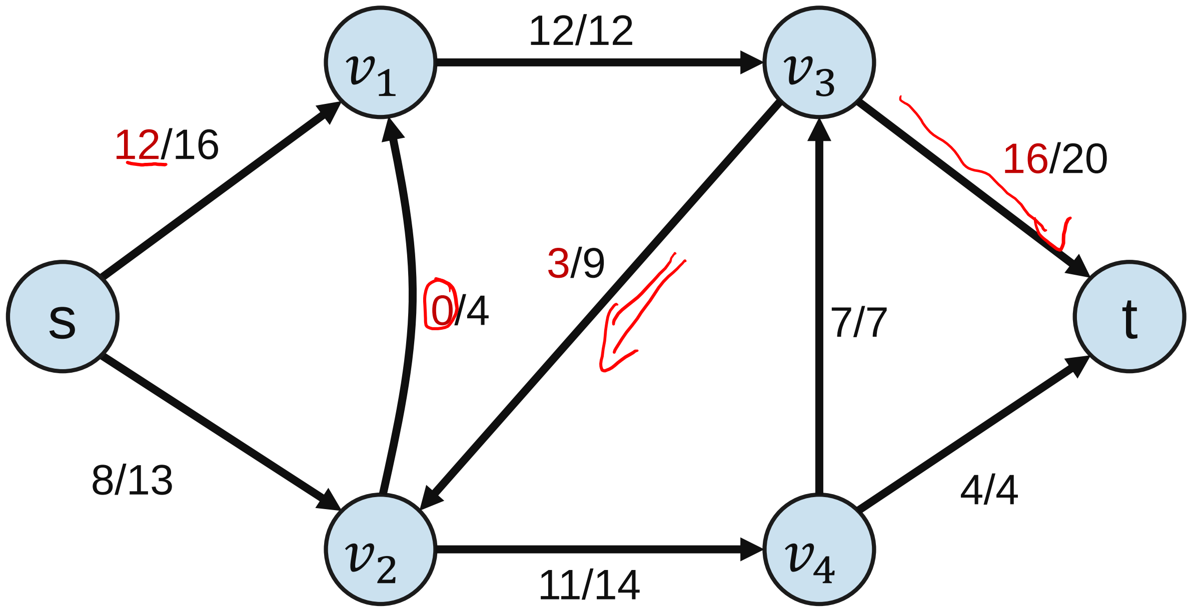




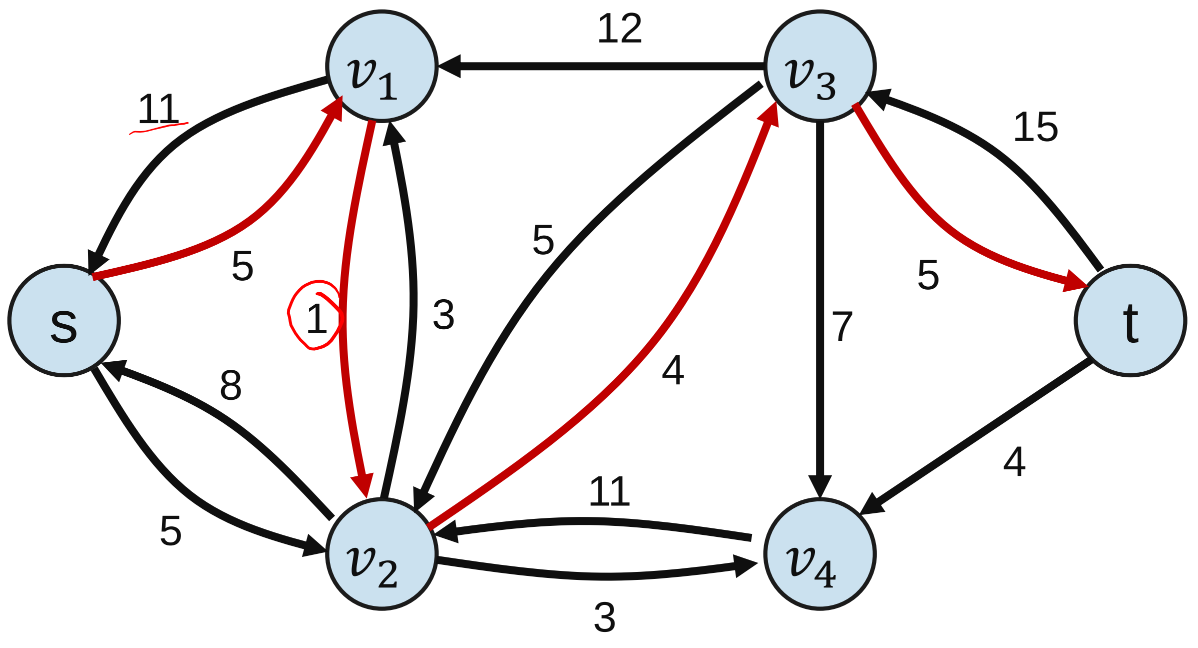
Example: Ford-Fulkerson-Method

flow network

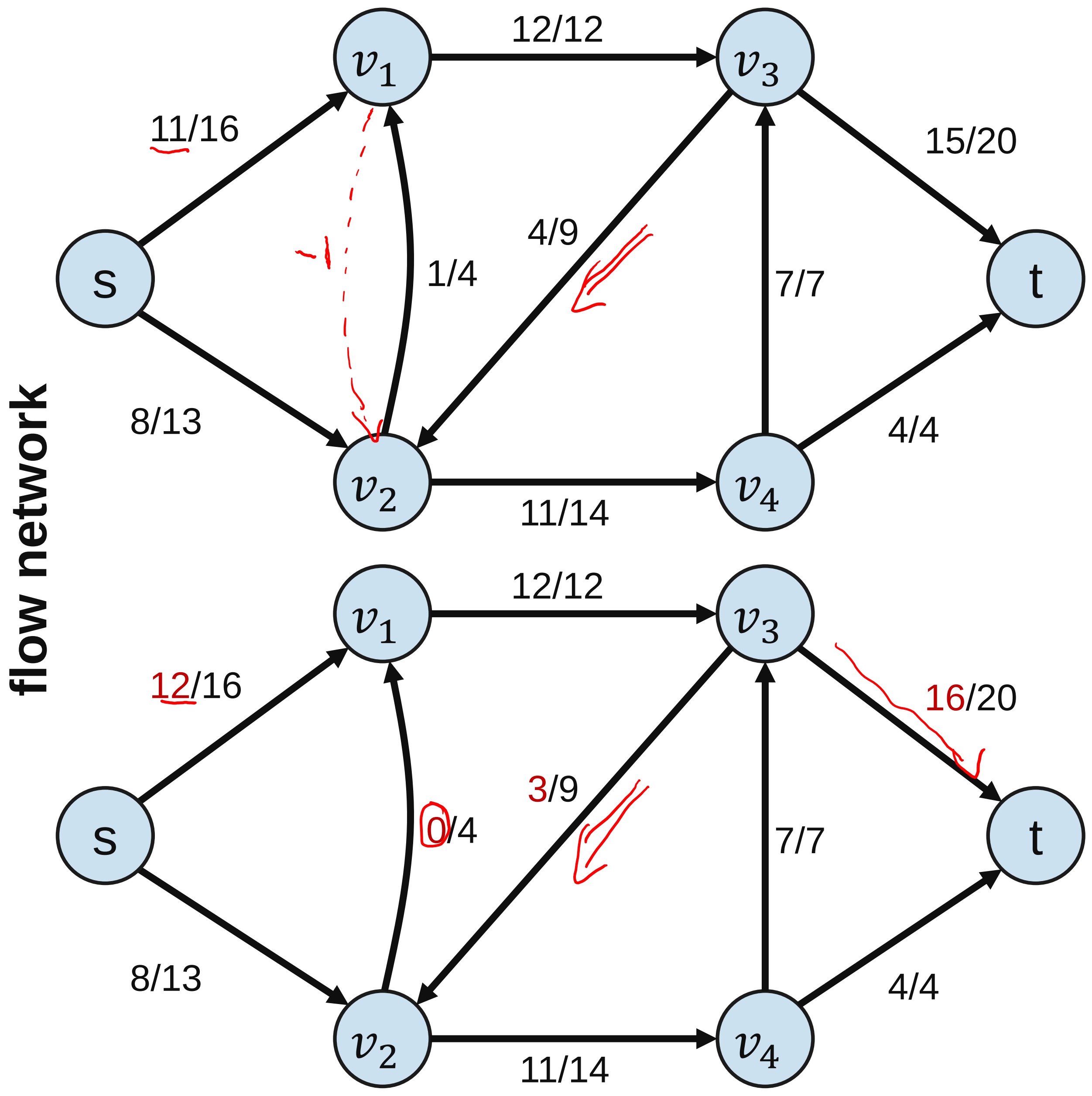


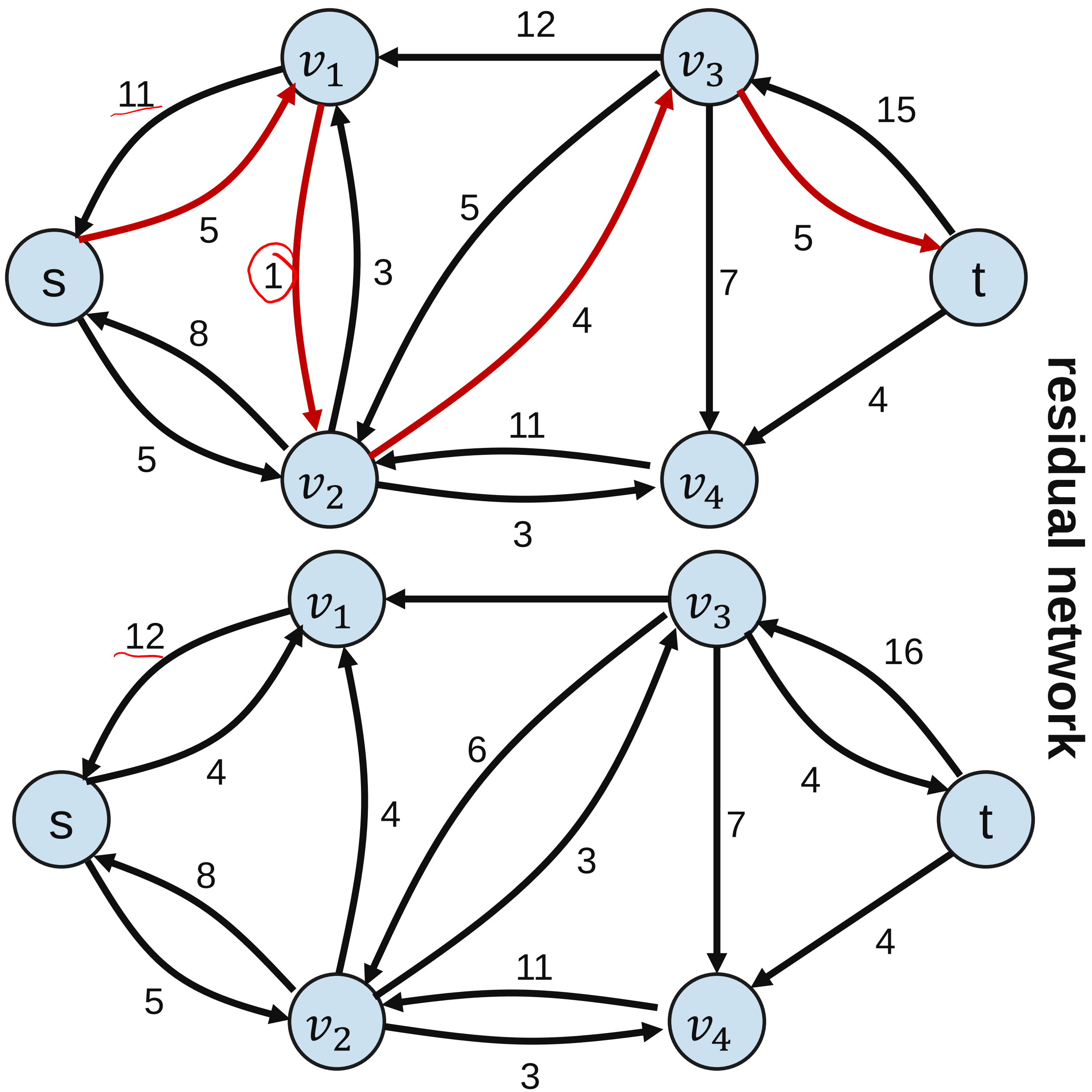


residual network



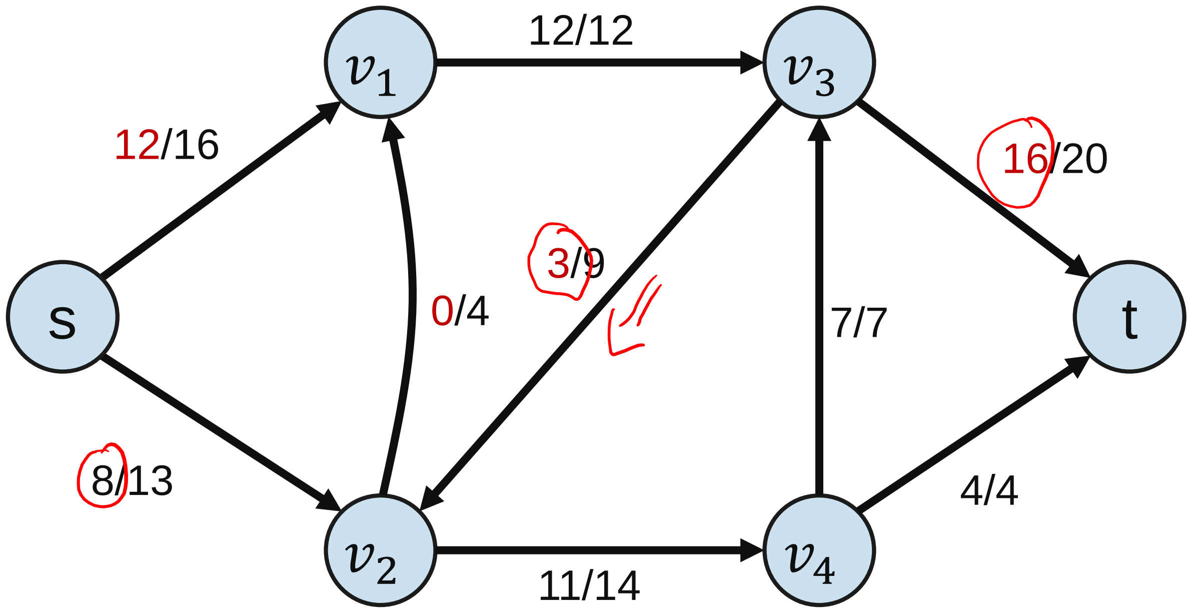
Example: Ford-Fulkerson-Method

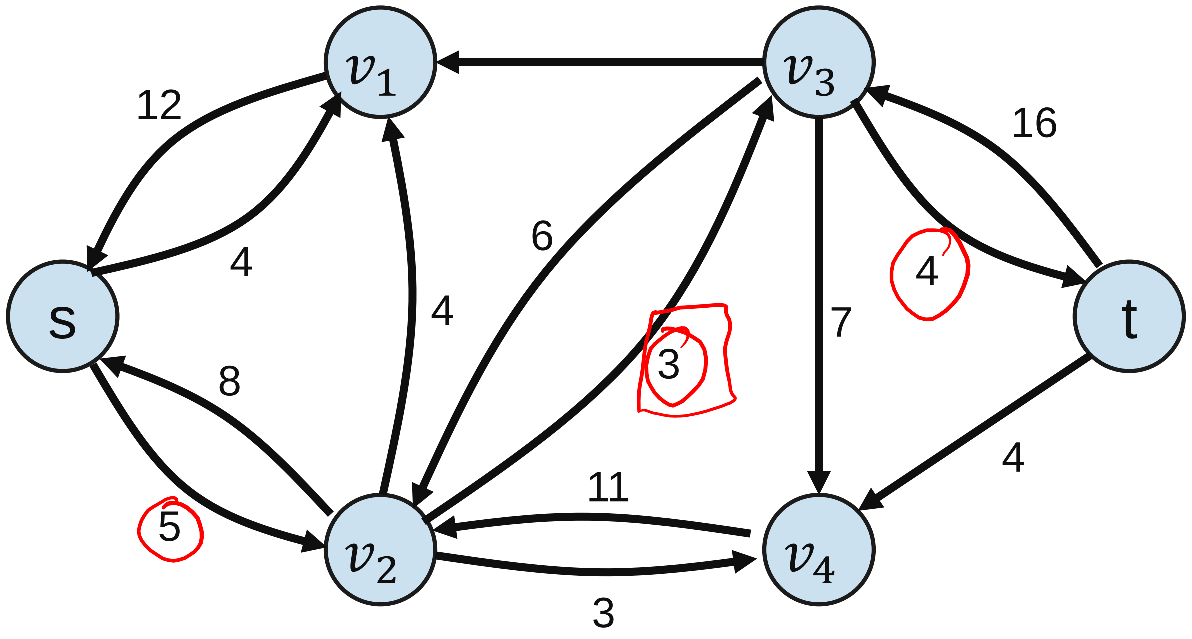




Example: Ford-Fulkerson-Method

flow network

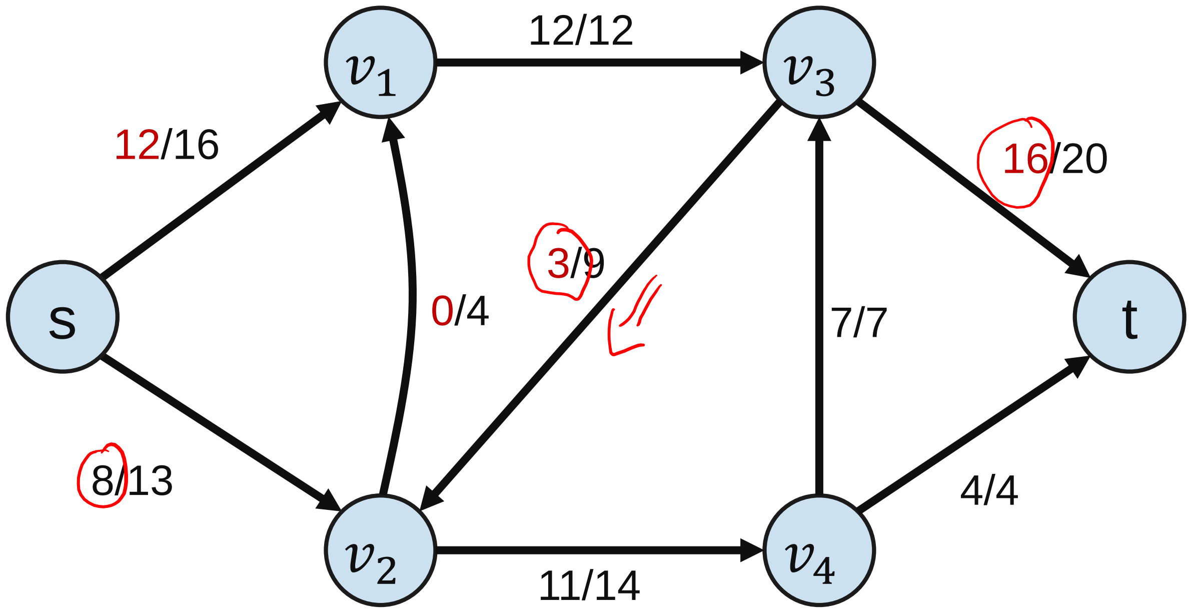


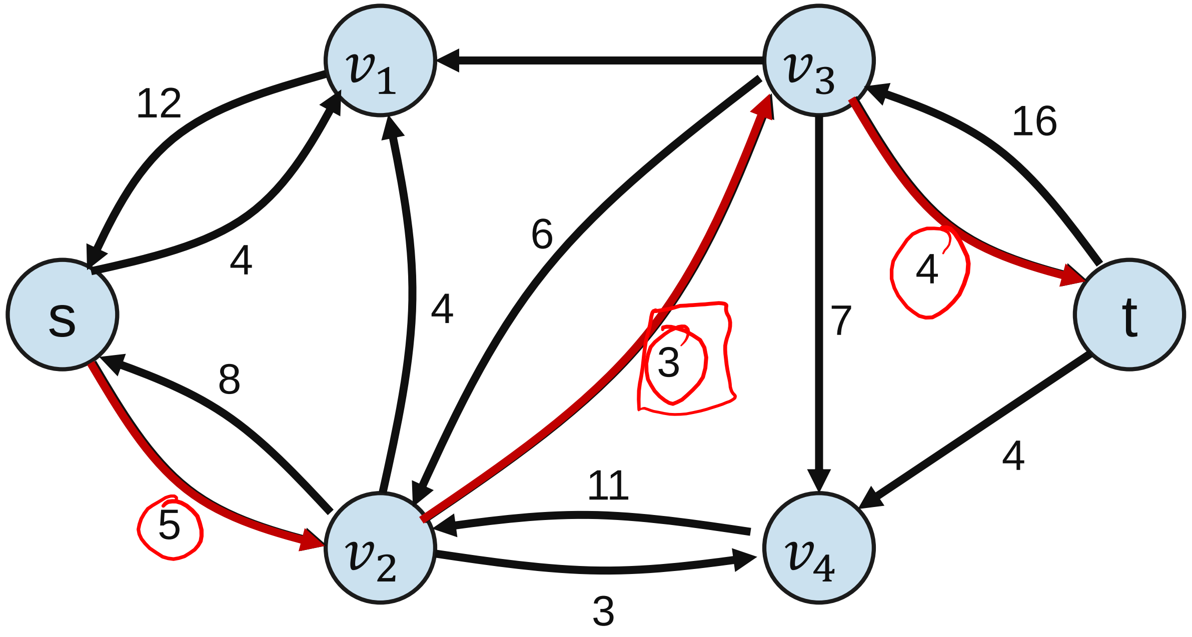


residual network

Example: Ford-Fulkerson-Method

flow network

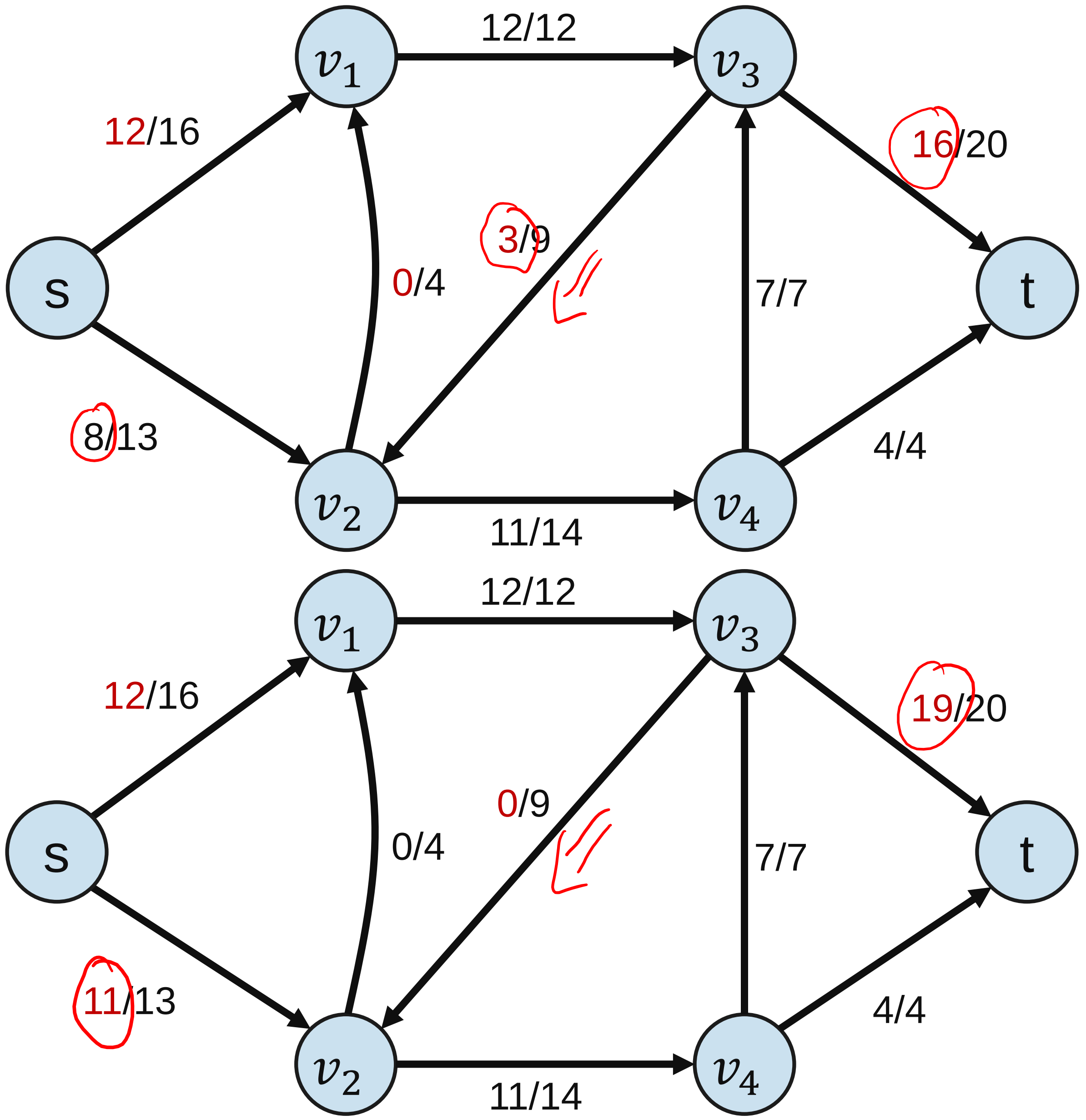




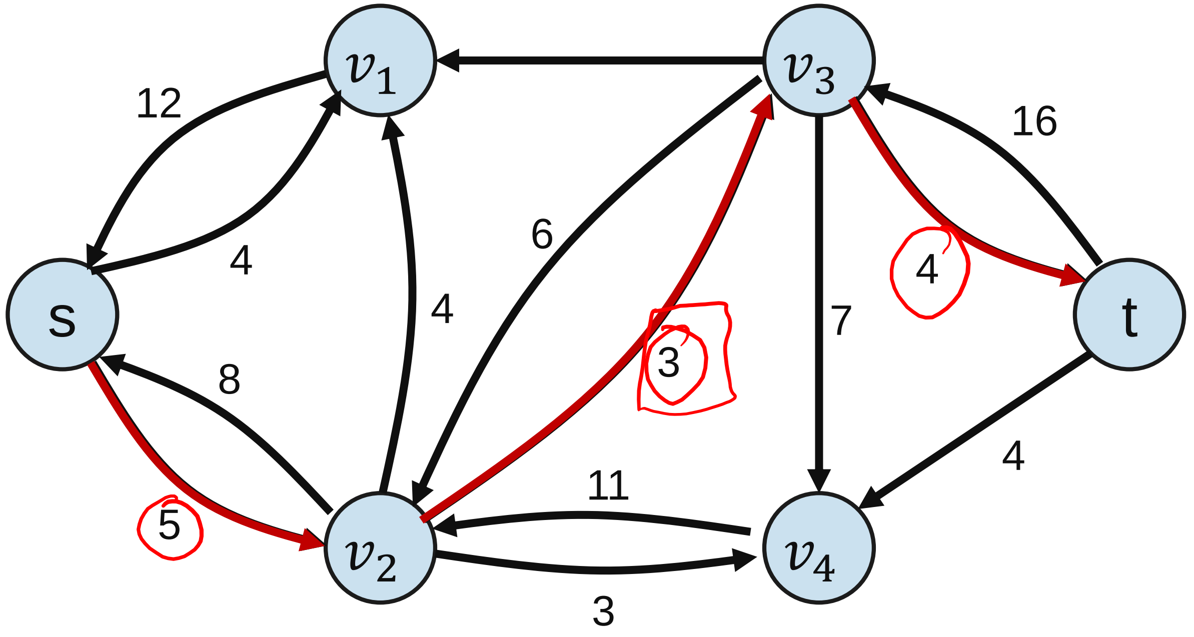
residual network

Example: Ford-Fulkerson-Method

flow network

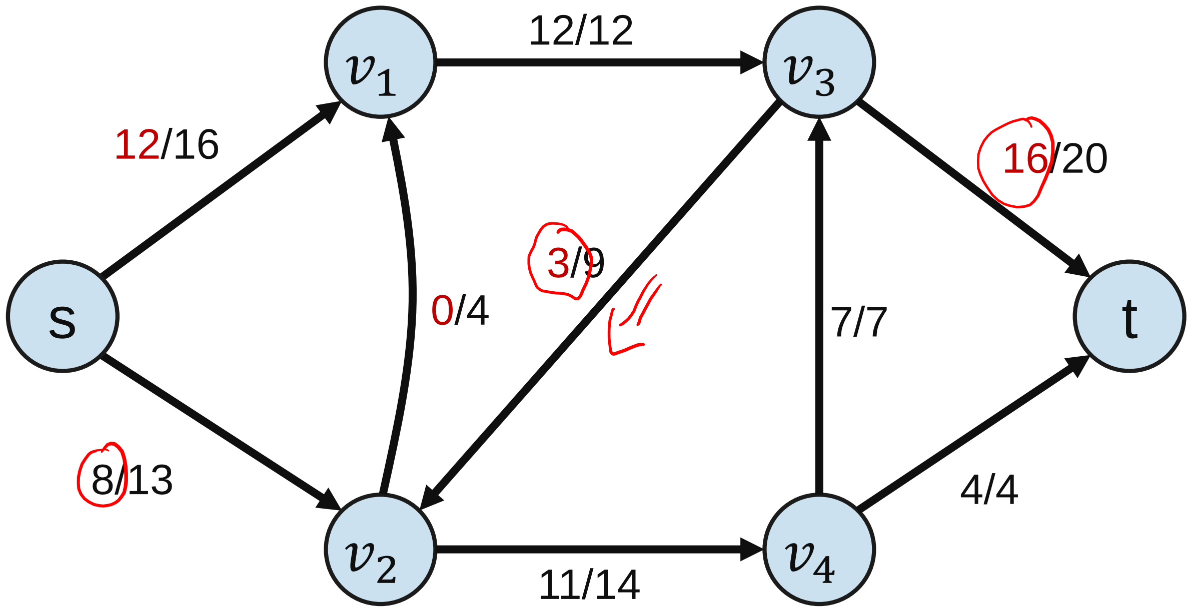


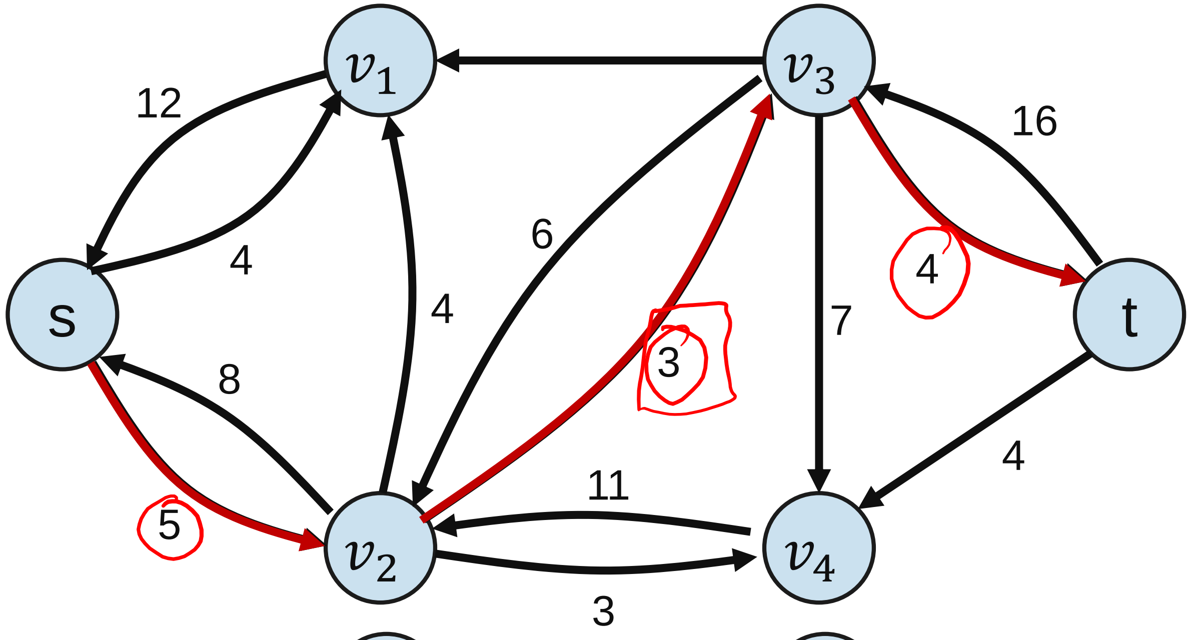
residual network

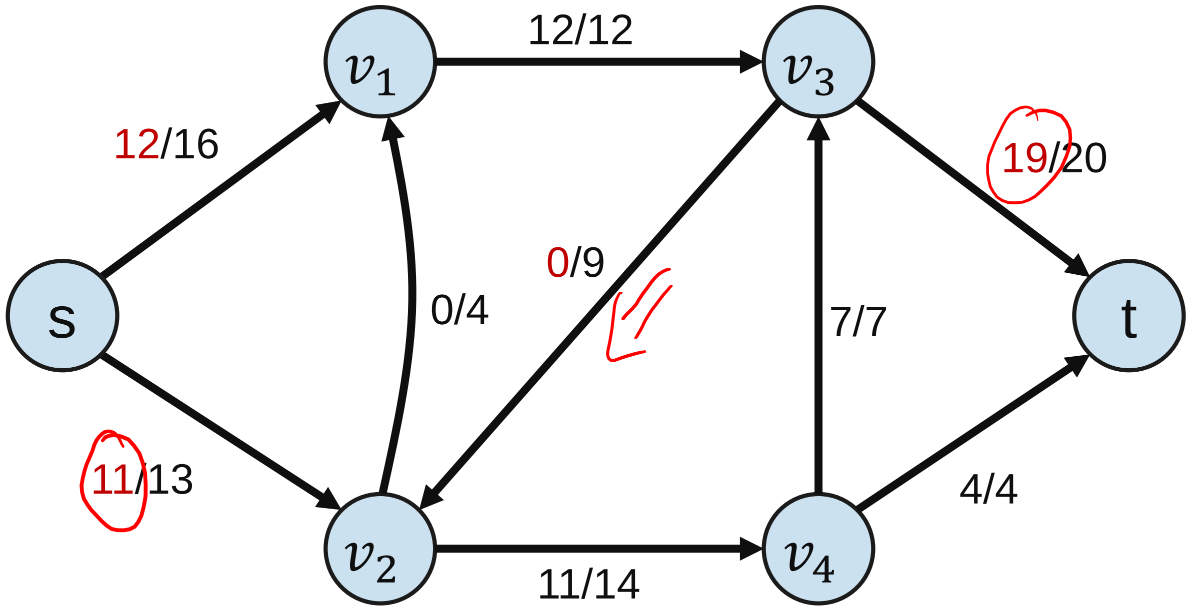


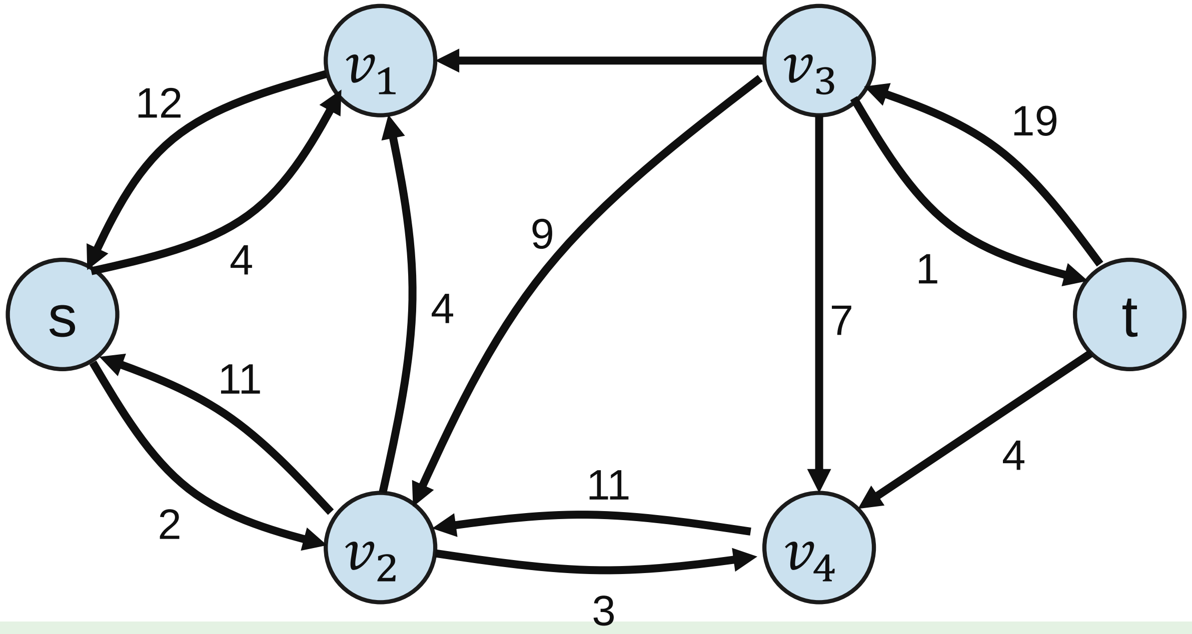
Example: Ford-Fulkerson-Method

flow network



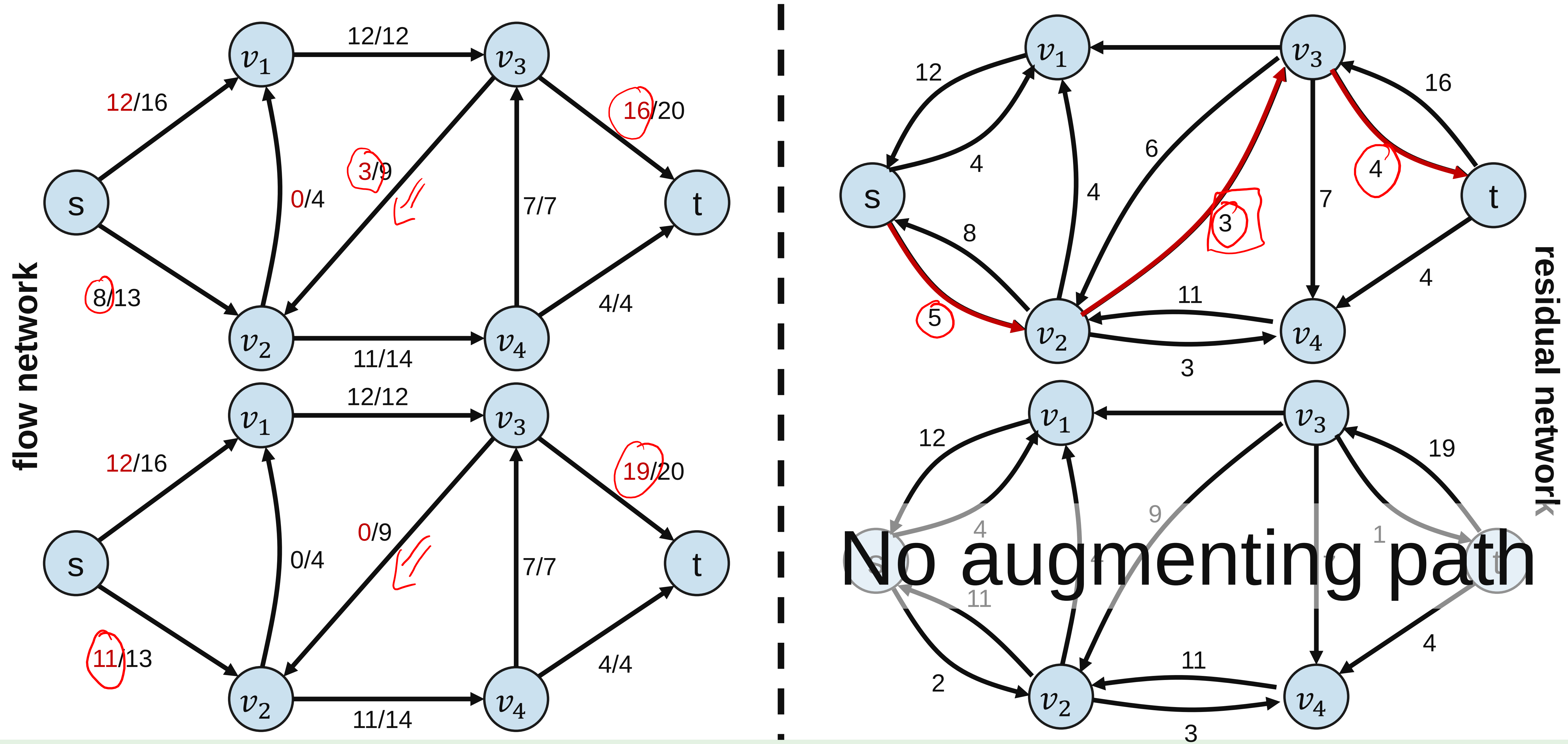






residual network

Example: Ford-Fulkerson-Method



Analysis 1: $f + f_p$ is a flow

Link to animated version: https://algorithms.discrete.ma.tum.de/

Correctness: $f + f_p$ is a flow

The augmented version is a flow:

Capacity constraint:

$$\begin{aligned}(f + f_p)(u,v) &= f(u,v) + f_p(u,v) \leq f(u,v) + c_f(u,v) \\ &\leq f(u,v) + \big(c(u,v) - f(u,v)\big) \qquad = c(u,v)\end{aligned}$$

(arguments use that that f_p is a flow in G_f and that f is a flow in G)

Correctness: $f + f_p$ is a flow

The augmented version is a flow:

Capacity constraint:

$$(f + f_p)(u,v) = f(u,v) + f_p(u,v) \le f(u,v) + c_f(u,v)$$
$$\le f(u,v) + \big(c(u,v) - f(u,v)\big) = c(u,v)$$

Conservation of flow:

Let $u \in V \setminus \{s,t\}$. $(f + f_p)(u,V) = f(u,V) + f_p(u,V) = 0 + 0 = 0$

(arguments use that that f_p is a flow in G_f and that f is a flow in G)

Correctness: $f + f_p$ is a flow

The augmented version is a flow:

Capacity constraint:

$$(f + f_p)(u,v) = f(u,v) + f_p(u,v) \leq f(u,v) + c_f(u,v)$$
$$\leq f(u,v) + \big(c(u,v) - f(u,v)\big) = c(u,v)$$

Conservation of flow:

Let $u \in V \setminus \{s,t\}$. $(f + f_p)(u,V) = f(u,V) + f_p(u,V) = 0 + 0 = 0$

Skew Symmetry:

$$(f + f_p)(u,v) = f(u,v) + f_p(u,v) = -f(v,u) - f_p(v,u)$$
$$= -(f(v,u) + f_p(v,u)) = -(f + f_p)(v,u)$$

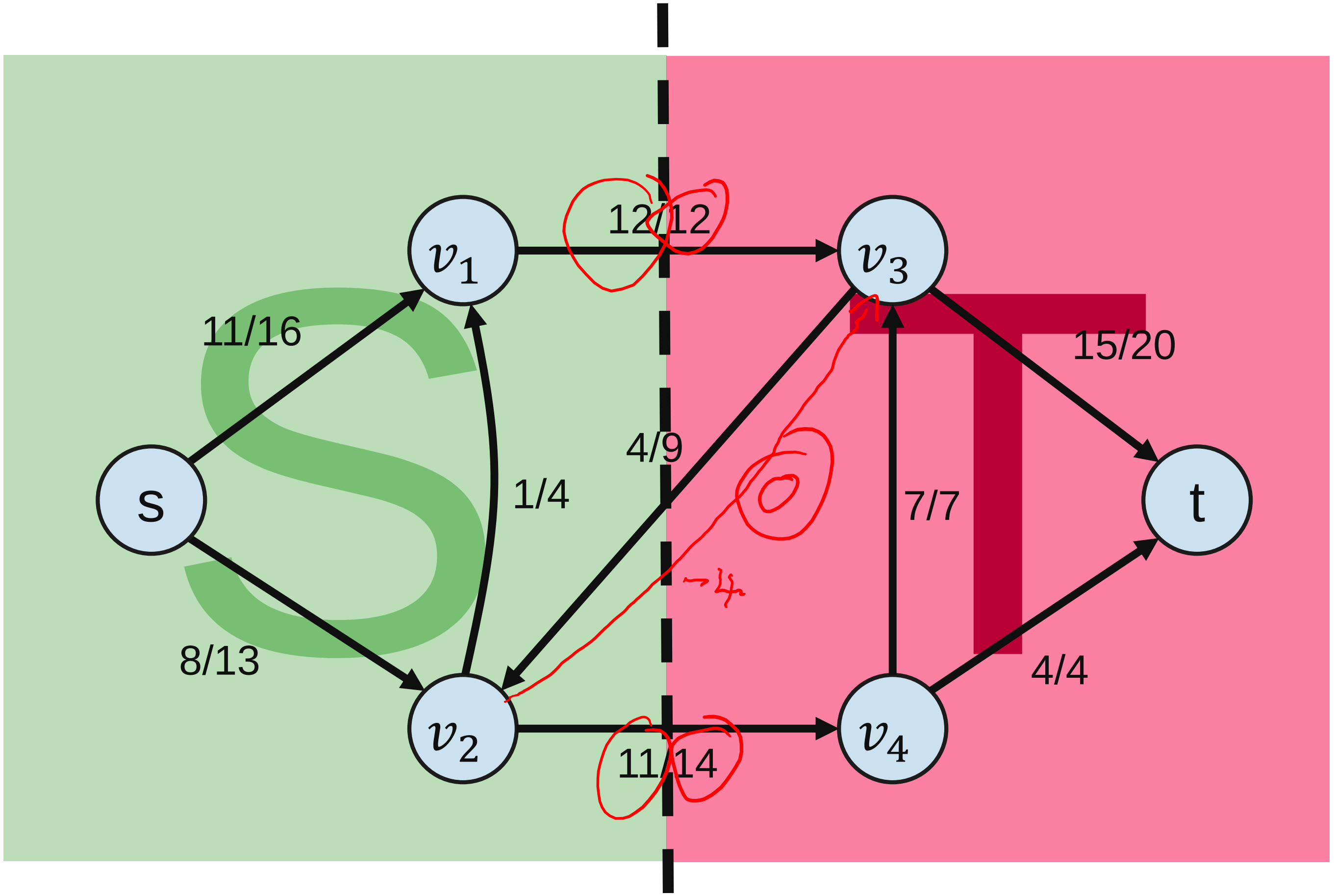
(arguments use that that f_p is a flow in G_f and that f is a flow in G)

Analysis 2: A maximum flow?

Min-cut=Max-flow

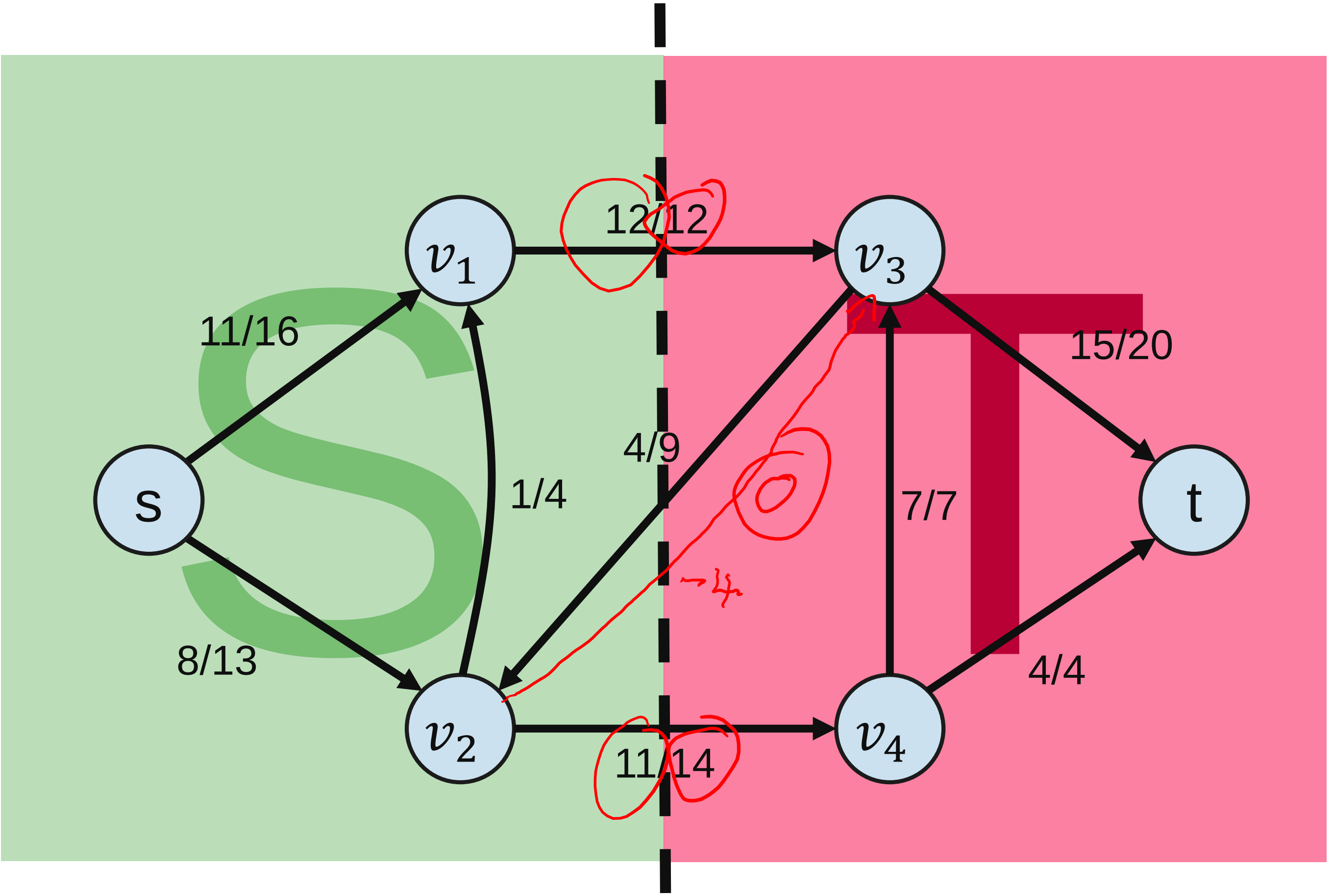
(seemingly)

An s-t-cut (S, T) in G is a partition of V ($V = S \cup T$, $S \cap T = \emptyset$) with $s \in S, t \in T$.



Cut

An s-t-cut (S,T) in G is a partition of V ($V = S \cup T$, $S \cap T = \emptyset$) with $s \in S, t \in T$.



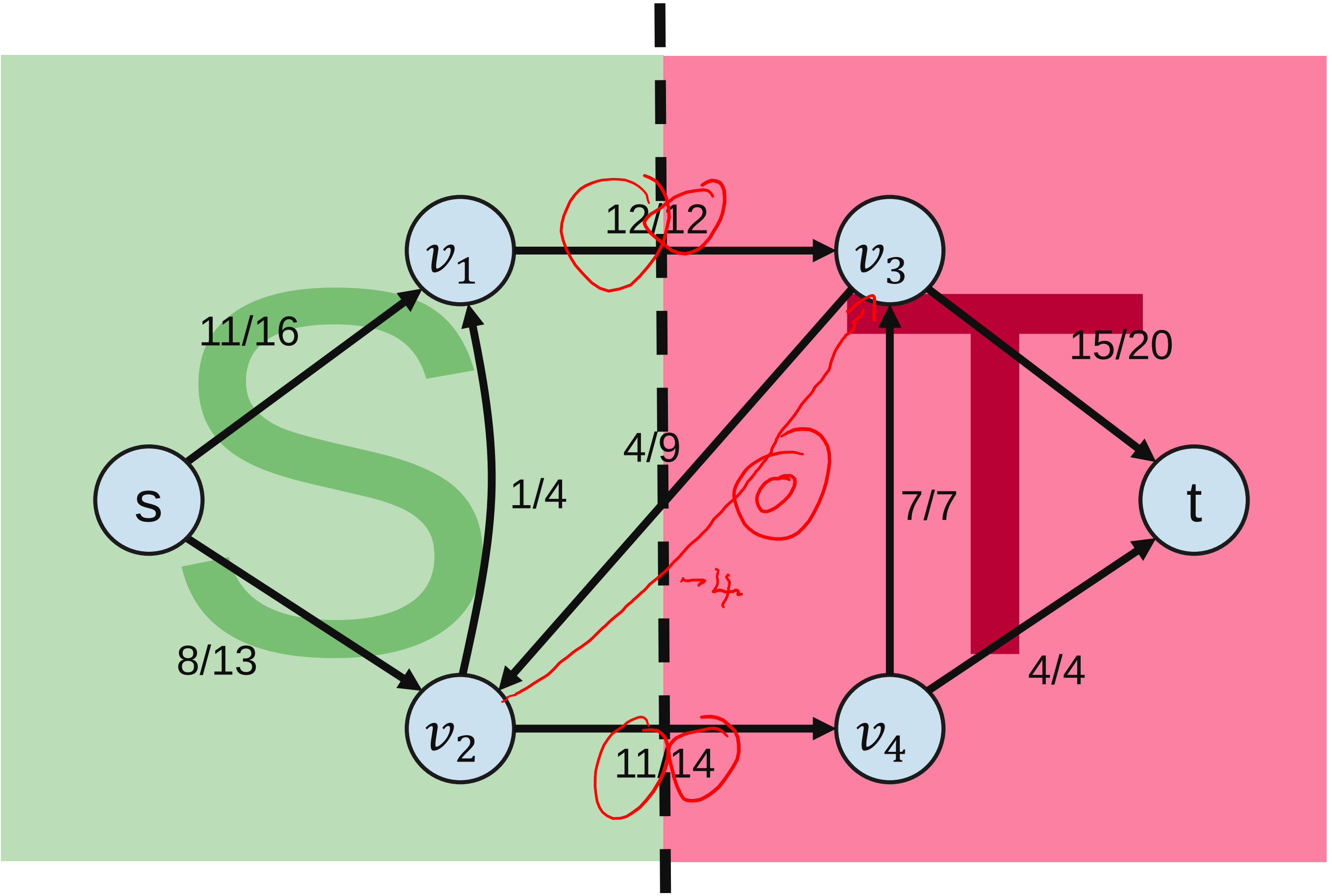
$c(S,T) = 26$

The **capacity** of a cut is

$$c(S,T) = \sum_{u \in S, v \in T} c(u,v)$$

Cut

An s-t-cut (S, T) in G is a partition of V ($V = S \cup T$, $S \cap T = \emptyset$) with $s \in S, t \in T$.



The **capacity** of a cut is

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v)$$

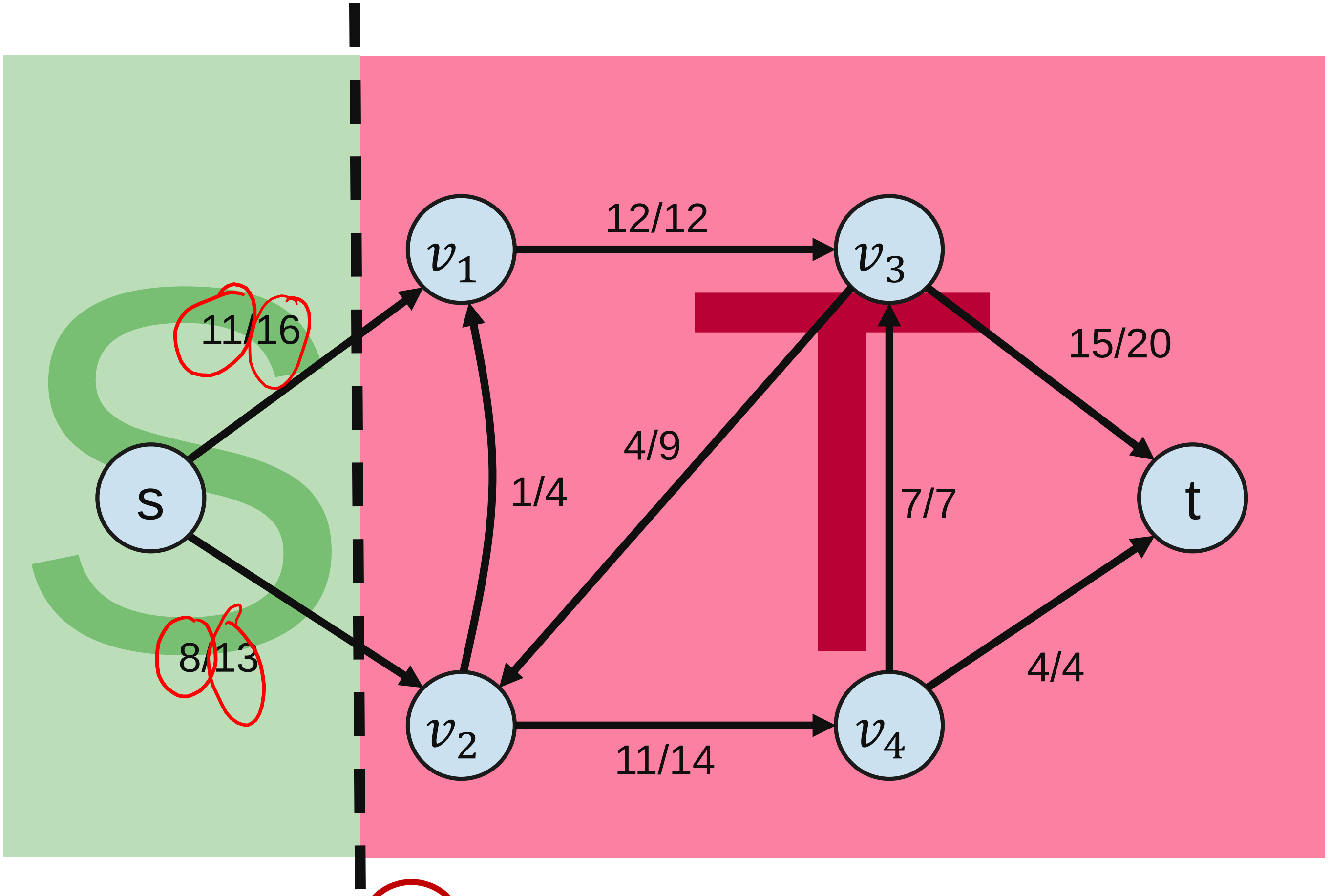
The **flow** over a cut is

$$f(S, T) = \sum_{u \in S, v \in T} f(u, v)$$

$c(S, T) = 26$

$f(S, T) = 12 + 11 + (-4) = 19$

An s-t-cut (S,T) in G is a partition of V ($V = S \cup T$, $S \cap T = \emptyset$) with $s \in S, t \in T$.



The **capacity** of a cut is

$$c(S,T) = \sum_{u \in S, v \in T} c(u,v)$$

The **flow** over a cut is

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v)$$

$c(S,T) = 29$

change!

$f(S,T) = \cancel{12 + 11 + (-4)} = 19$

$11 + 8$

no change!

Lemma A: For each flow f and each cut (S,T) we have $|f| = f(S,T)$.

s

$= f(s,V) + \sum_{u \in S \setminus \{s\}} f(u,V)$

Def

Lemma A: For each flow f and each cut (S,T) we have $|f| = f(S,T)$.

s

$= f(s,V) + \sum_{u \in S \setminus \{s\}} f(u,V)$

Def

Notation:

$$f(x,Y) = \sum_{y \in Y} f(x,y)$$

$$f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$$

Lemma A: For each flow f and each cut (S,T) we have $|f| = f(S,T)$.

Proof:

$$f(S,T) = f(S,V) - f(S,\overbrace{V \setminus T}^{=S})$$

$$= f(S,V) - f(S,S) = f(S,V) = f(s,V) + \sum_{u \in S\setminus\{s\}} f(u,V)$$

$$= f(s,V) + f(S \setminus \{s\}, V) = f(s,V) \overset{\text{Def}}{=} |f|$$

Notation:

$$f(x,Y) = \sum_{y \in Y} f(x,y)$$

$$f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$$

Lemma A: For each flow f and each cut (S,T) we have $|f| = f(S,T)$.

Proof:

insert definition

(omitted simple proof, using symmetry)

$f(S,S) = 0$

$$f(S,T) = f(S,V) - f(S,\overset{S}{V \setminus T})$$

$$= f(S,V) - f(S,S) = f(S,V) = f(s,V) + \sum_{u \in S \setminus \{s\}} f(u,V)$$

$$= f(s,V) + \underline{f(S \setminus \{s\},V)} = f(s,V) \overset{Def}{=} |f|$$

$\underline{V \setminus T = S}$

$f(S \setminus \{s\},V) = 0$ as
$f(u,V) = 0$ for all $u \neq s,t$
due to flow conservation

Notation:

$$f(x,Y) = \sum_{y \in Y} f(x,y)$$

$$f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$$

Max-Flow = Min-Cut Theorem

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Corollaries:

- If the Ford-Fulkerson-Method terminates, it has computed a maximum flow
- The capacity of the "smallest cut" equals the capacity of the maximum flow

Max-Flow = Min-Cut Theorem: Proof **1→ 2**

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 1→ 2:

Let f be a maximum flow.

Assume there is an augmenting path p in G_f.

Then $f + f_p$ is a flow with $|f + f_p| > |f|$, a contradiction.

Max-Flow = Min-Cut Theorem: Proof **1→ 2**

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 1→ 2:

Let f be a maximum flow.

Assume there is an augmenting path p in G_f.

Then $f + f_p$ is a flow with $|f + f_p| > |f|$, a contradiction.

Adding an augmenting path flow f_p always increases the value of the flow. Why?

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 2→ 3:

Assume that G_f has no s-t path. Define:

$S := \{ v \in V \mid$ there is an s-v path in $G_f\}$, $T := V \setminus S$ (note $T \neq \emptyset$)

Max-Flow = Min-Cut Theorem: Proof 2→ 3

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

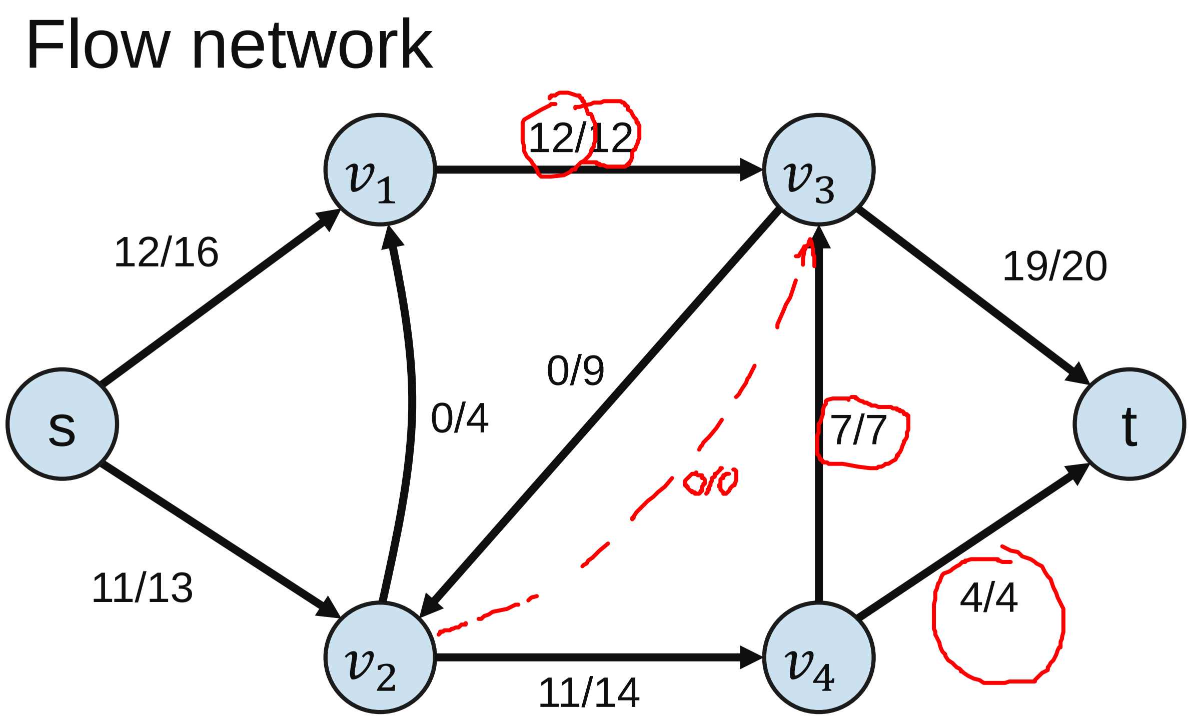
Proof 2→ 3:

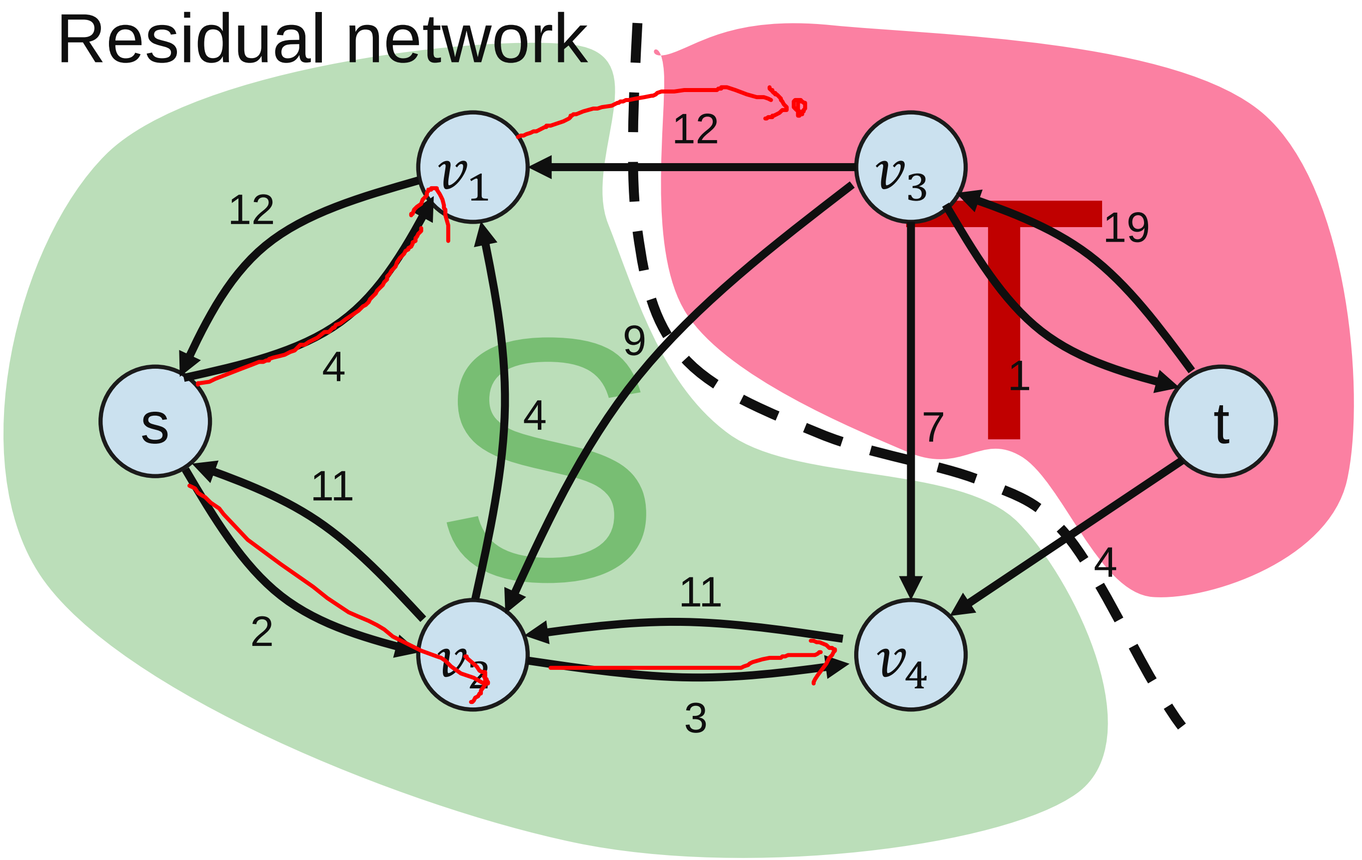
Assume that G_f has no s-t path. Define:

$S := \{ v \in V \mid \text{there is an } s\text{-}v \text{ path in } G_f\}$, $T := V \setminus S$ (note $T \neq \emptyset$)

Then (S, T) is a cut and we have $f(u, v) = c(u, v)$ for all $u \in S, v \in T$.

$$|f| \underset{\substack{\uparrow \\ \text{Lemma A}}}{=} f(S, T) = \sum_{u \in S, v \in T} f(u, v) = \sum_{u \in S, v \in T} c(u, v) = c(S, T).$$





Proof 2→ 3:

Assume that G_f has no s-t path. Define:

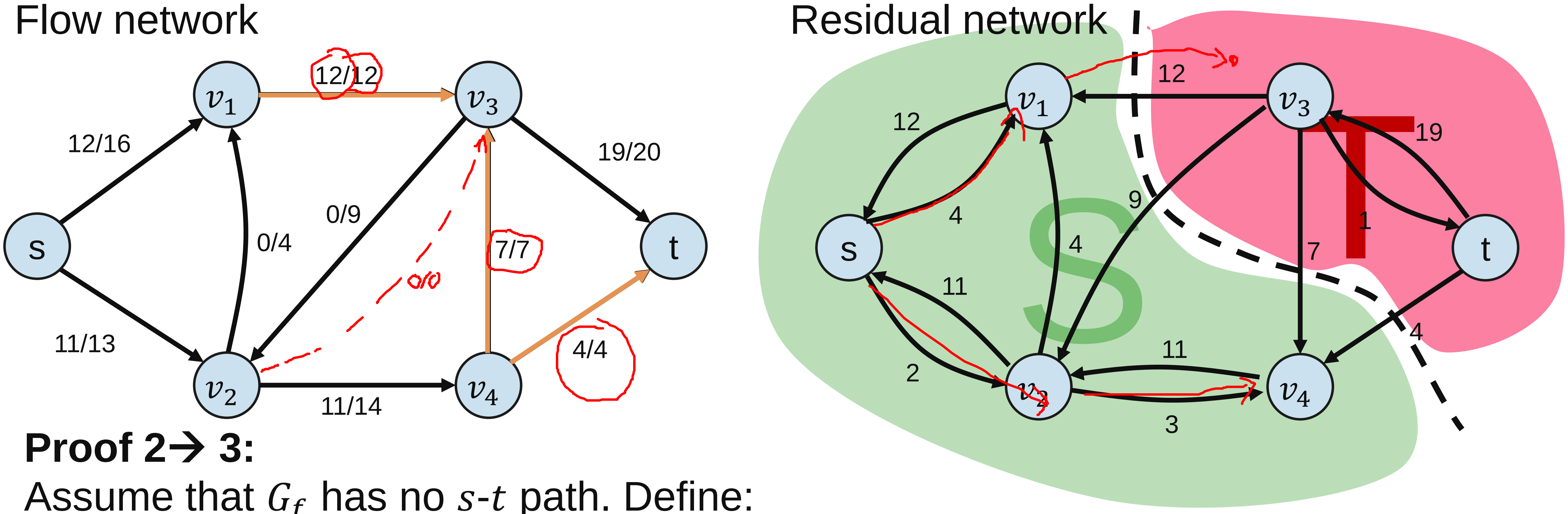
$S := \{ v \in V \mid$ there is an s-v path in $G_f\}$, $T := V \setminus S$ (note $T \neq \emptyset$)

Then (S,T) is a cut and we have $f(u,v) = c(u,v)$ for all $u \in S, v \in T$.

$$|f| = f(S,T) = \sum_{u \in S, v \in T} f(u,v) = \sum_{u \in S, v \in T} c(u,v) = c(S,T).$$

Max-Flow = Min-Cut Theorem: Proof 2→ 3

Flow network   Residual network



Proof 2→ 3:

Assume that G_f has no s-t path. Define:

$S := \{ v \in V \mid$ there is an s-v path in $G_f\}$, $T := V \setminus S$ (note $T \neq \emptyset$)

Then (S, T) is a cut and we have $f(u, v) = c(u, v)$ for all $u \in S, v \in T$.

$$|f| = f(S, T) = \sum_{u \in S, v \in T} f(u, v) = \sum_{u \in S, v \in T} c(u, v) = c(S, T).$$

Max-Flow = Min-Cut Theorem: Proof **3→ 1**

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 3→ 1:

Pick the cut (S, T) from 3 and an arbitrary flow f:

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 3→ 1:

Pick the cut (S, T) from 3 and an arbitrary flow f:

Lemma A

$$|f| = f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v)$$

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 3→ 1:

Pick the cut (S, T) from 3 and an arbitrary flow f:

$$\underbrace{|f| = f(S,T)}_{\text{Lemma A}} = \sum_{u \in S} \sum_{v \in T} f(u,v) \leq \sum_{u \in S} \sum_{v \in T} c(u,v) = c(S,T)$$

Theorem: Let f be an s-t-flow in a flow network $(G = (V, E), c)$.

The following are equivalent

1. flow f is a maximum flow
2. The residual network G_f contains **no** augmenting path
3. $|f| = c(S, T)$ for a cut (S, T)

Proof 3→ 1:

Pick the cut (S, T) from 3 and an arbitrary flow f:

Lemma A

$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) \leq \sum_{u \in S} \sum_{v \in T} c(u,v) = c(S,T)$$

Thus, the value of every flow is at most $c(S, T)$, so, if it equals $c(S, T)$, it has to be maximal!

Theorem: If all capacities are integral, then the Ford-Fulkerson-method finds a maximum flow f after at most at most f^* (value of a maximum flow) augmentations. Further, all values of $f(u,v)$ are integral.

Theorem: If all capacities are integral, then the Ford-Fulkerson-method finds a maximum flow f after at most at most f^* (value of a maximum flow) augmentations. Further, all values of $f(u,v)$ are integral.

Integrality sounds innocent but it is crucial for many applications.

Theorem: If all capacities are integral, then the Ford-Fulkerson-method finds a maximum flow f after at most at most f^* (value of a maximum flow) augmentations. Further, all values of $f(u,v)$ are integral.

Integrality sounds innocent but it is crucial for many applications.

Proof:

- $|f|$ increases by at least one in each iteration → at most f^* iterations.

Theorem: If all capacities are integral, then the Ford-Fulkerson-method finds a maximum flow f after at most at most f^* (value of a maximum flow) augmentations. Further, all values of $f(u,v)$ are integral.

Integrality sounds innocent but it is crucial for many applications.

Proof:

- $|f|$ increases by at least one in each iteration → at most f^* iterations.
- By induction: The capacity of each augmenting path is integral, and $f(u,v)$ remains integral.

Theorem: If all capacities are integral, then the Ford-Fulkerson-method finds a maximum flow f after at most at most f^* (value of a maximum flow) augmentations. Further, all values of $f(u,v)$ are integral.

Integrality sounds innocent but it is crucial for many applications.

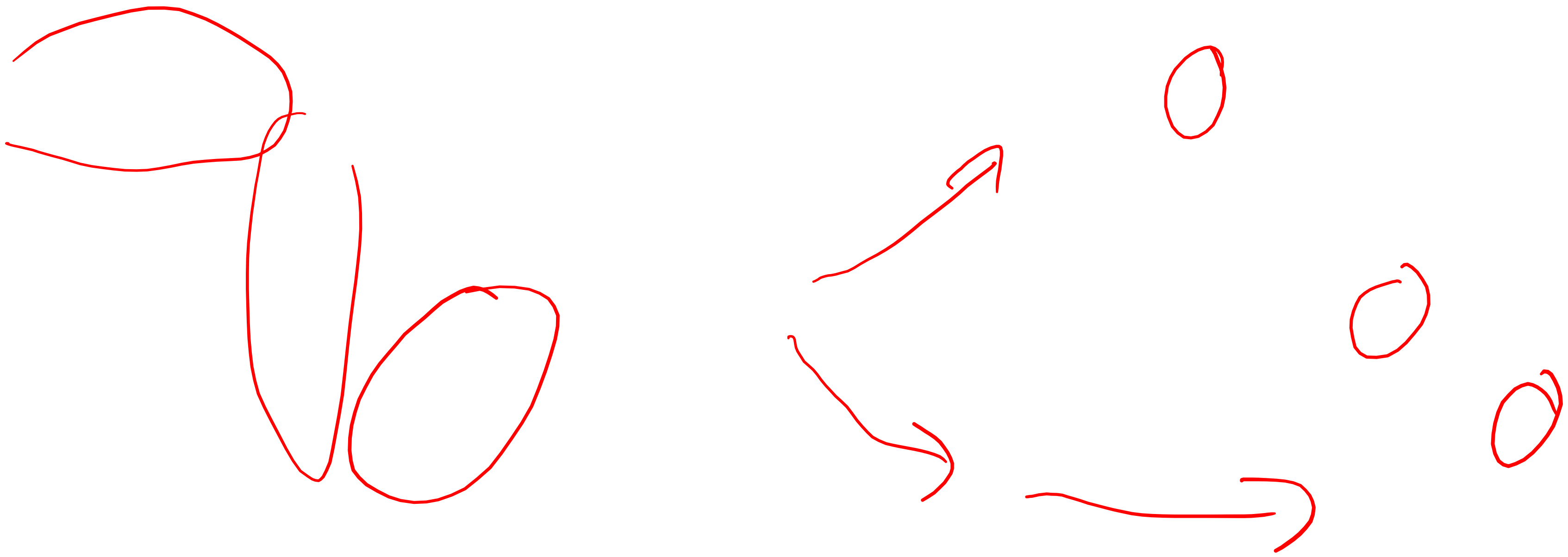
Proof:

- $|f|$ increases by at least one in each iteration → at most f^* iterations.
- By induction: The capacity of each augmenting path is integral, and $f(u,v)$ remains integral.

Warning! *The theorem statement does not hold for real-valued capacities.*
The ford-Fulkerson-Method might not even terminate!

How to find a min-cut?

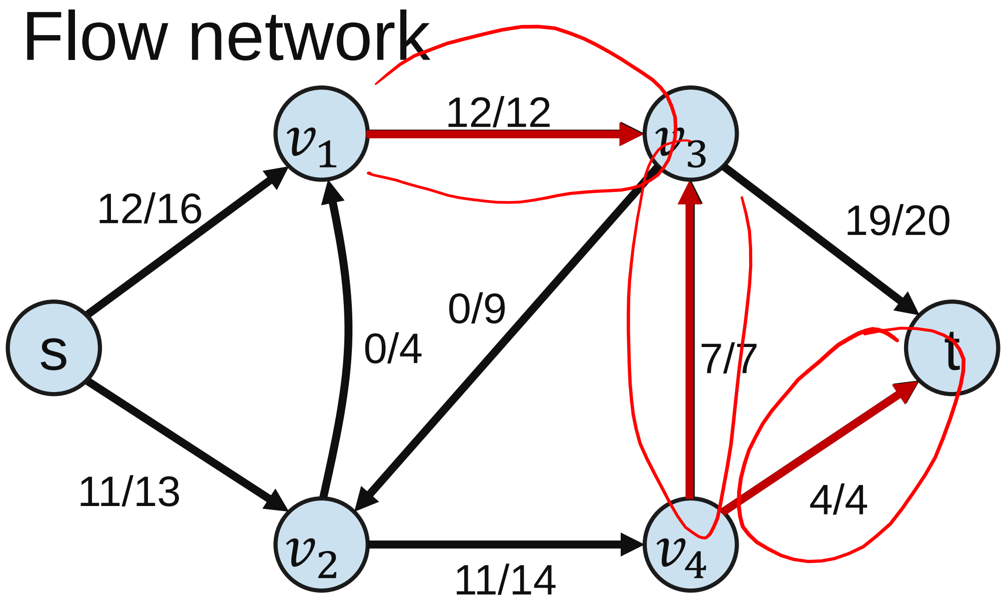
1. Compute a max-flow
2. Let S consist of the vertices that one can reach from s via non-critical edges (An edge (u, v) is critical if $c(u, v) = f(u, v)$)
3. Let $T = V \setminus S$.

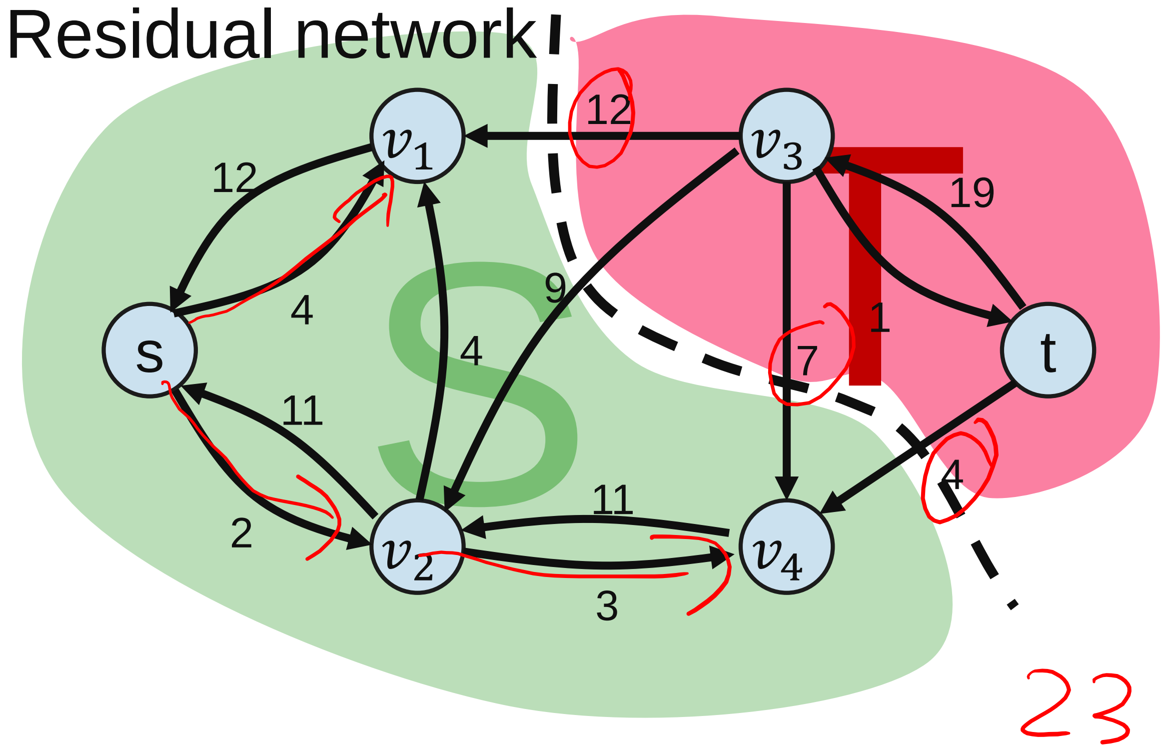


23

How to find a min-cut?

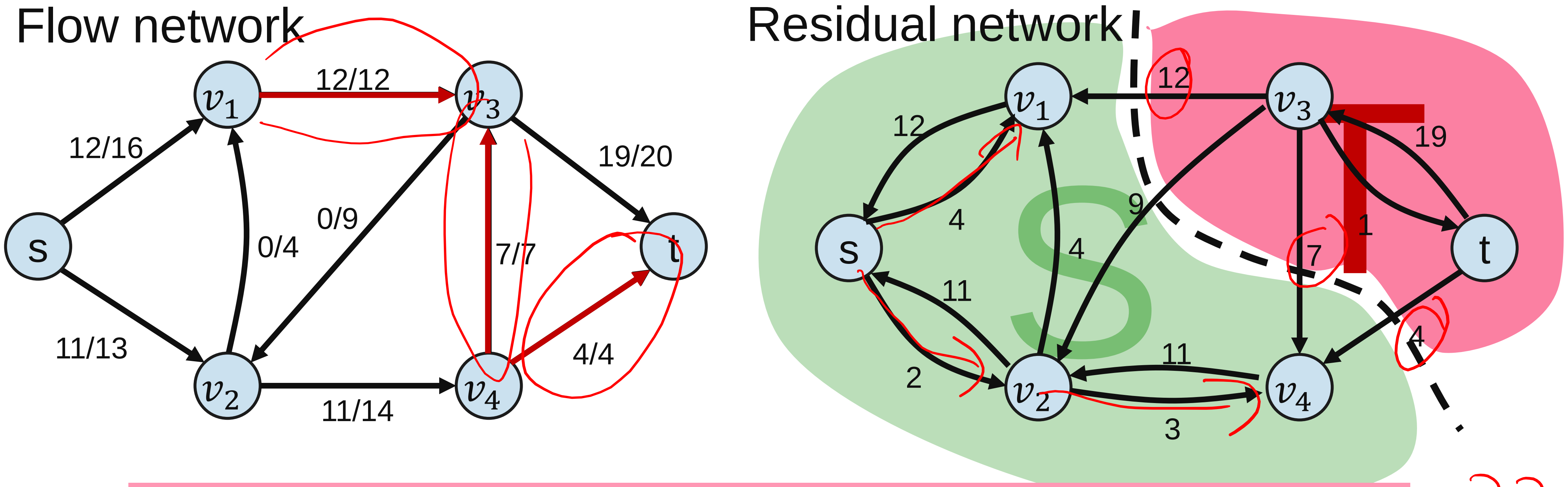
1. Compute a max-flow
2. Let S consist of the vertices that one can reach from s via non-critical edges (An edge (u, v) is critical if $c(u, v) = f(u, v)$)
3. Let $T = V \setminus S$.





How to find a min-cut?

1. Compute a max-flow
2. Let S consist of the vertices that one can reach from s via non-critical edges (An edge (u, v) is critical if $c(u, v) = f(u, v)$)
3. Let $T = V \setminus S$.



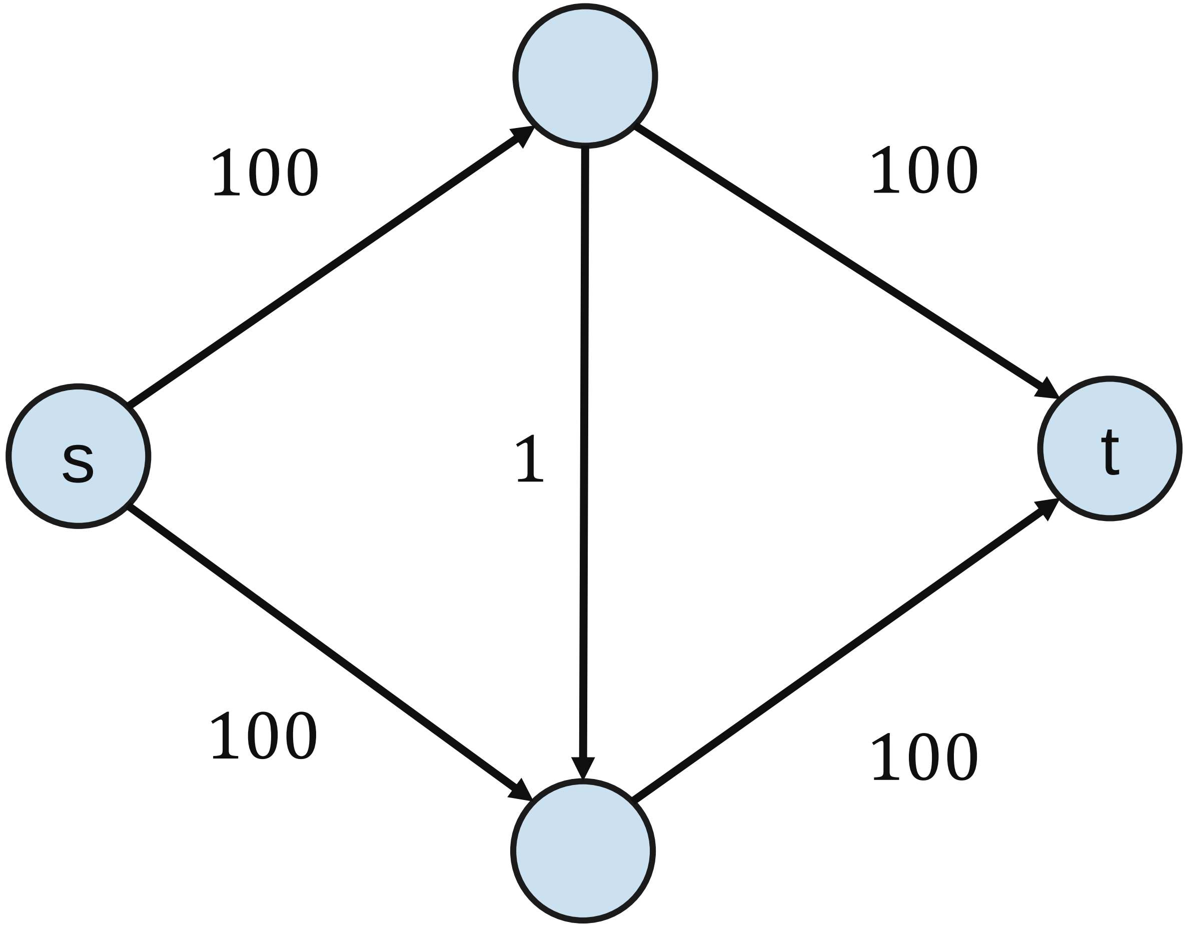
23

There are better, more direct methods! (not in this lecture)

Analysis 3: Runtime

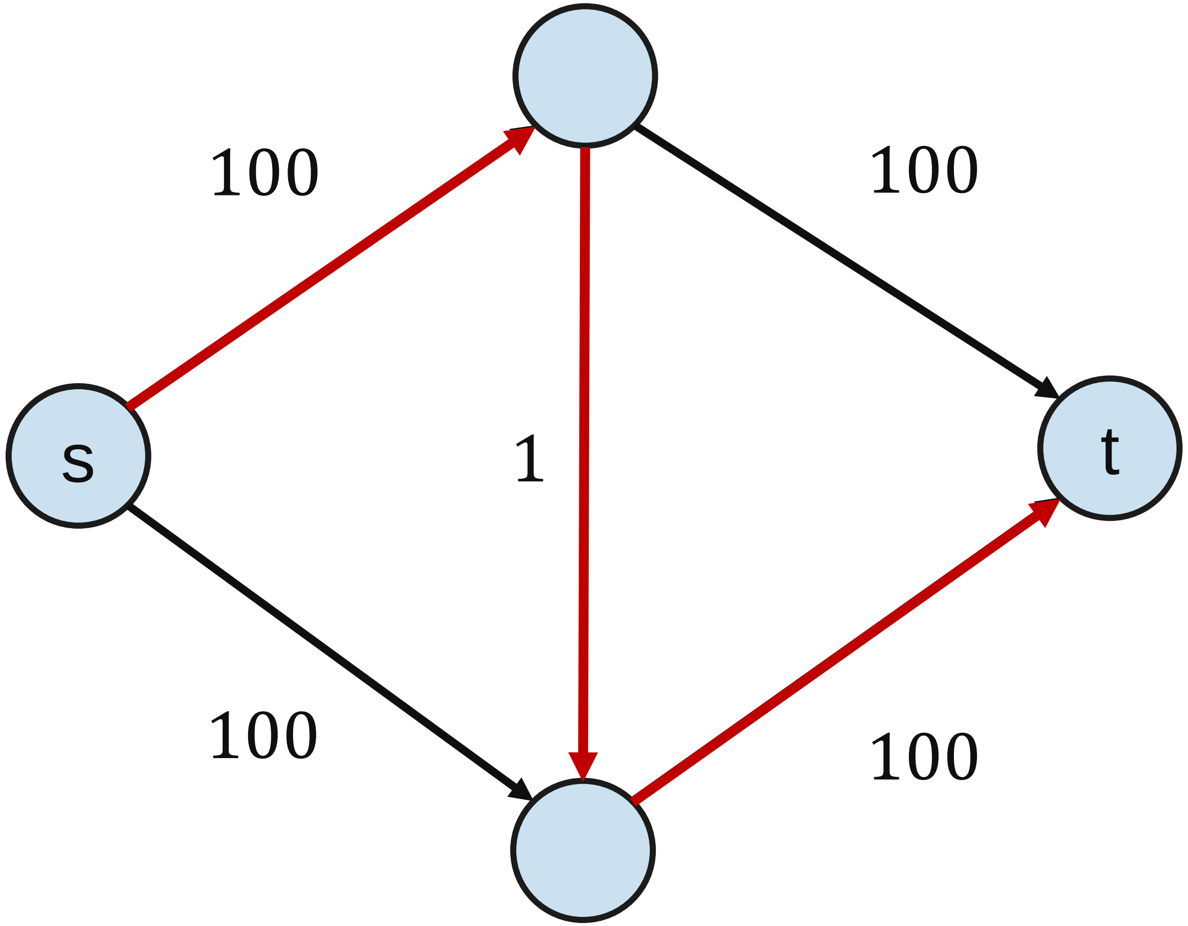
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



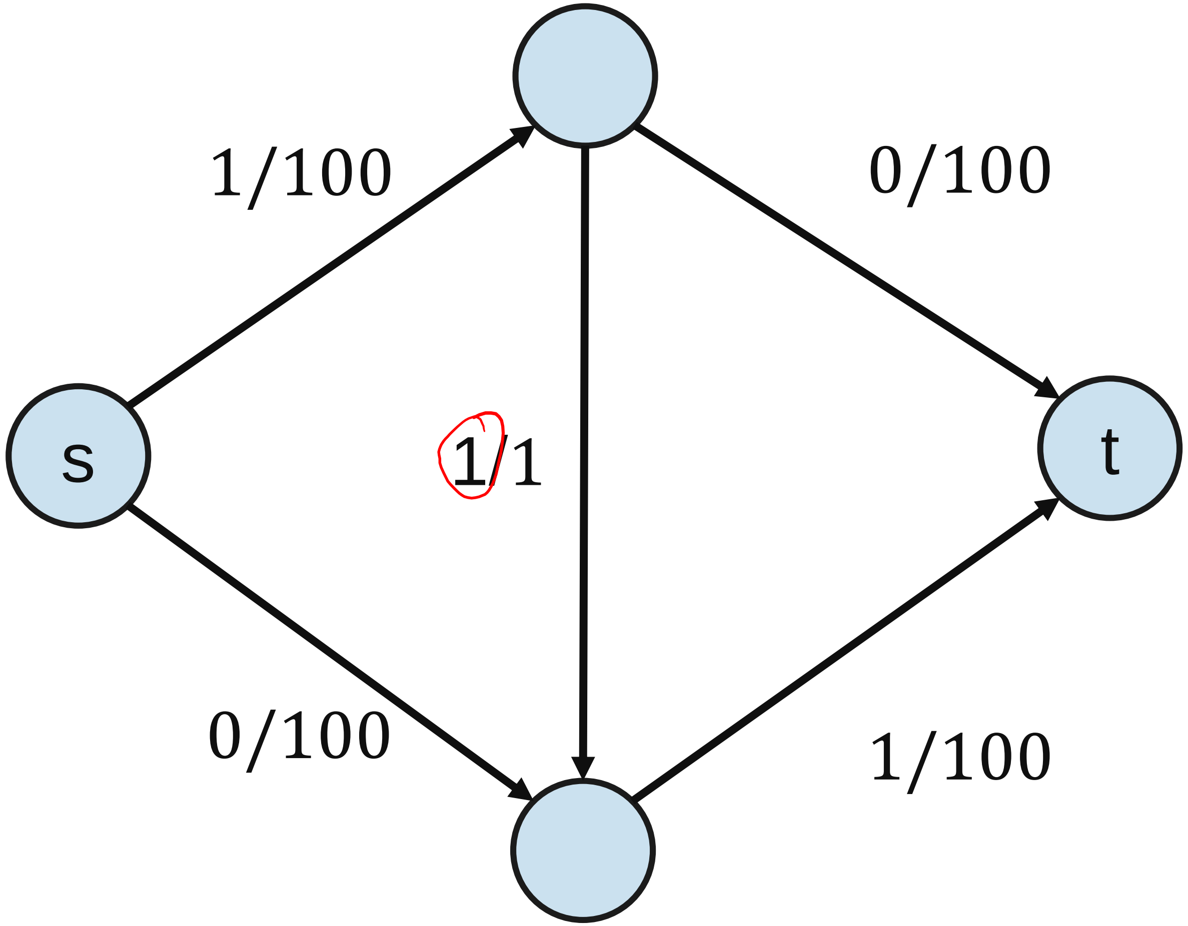
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



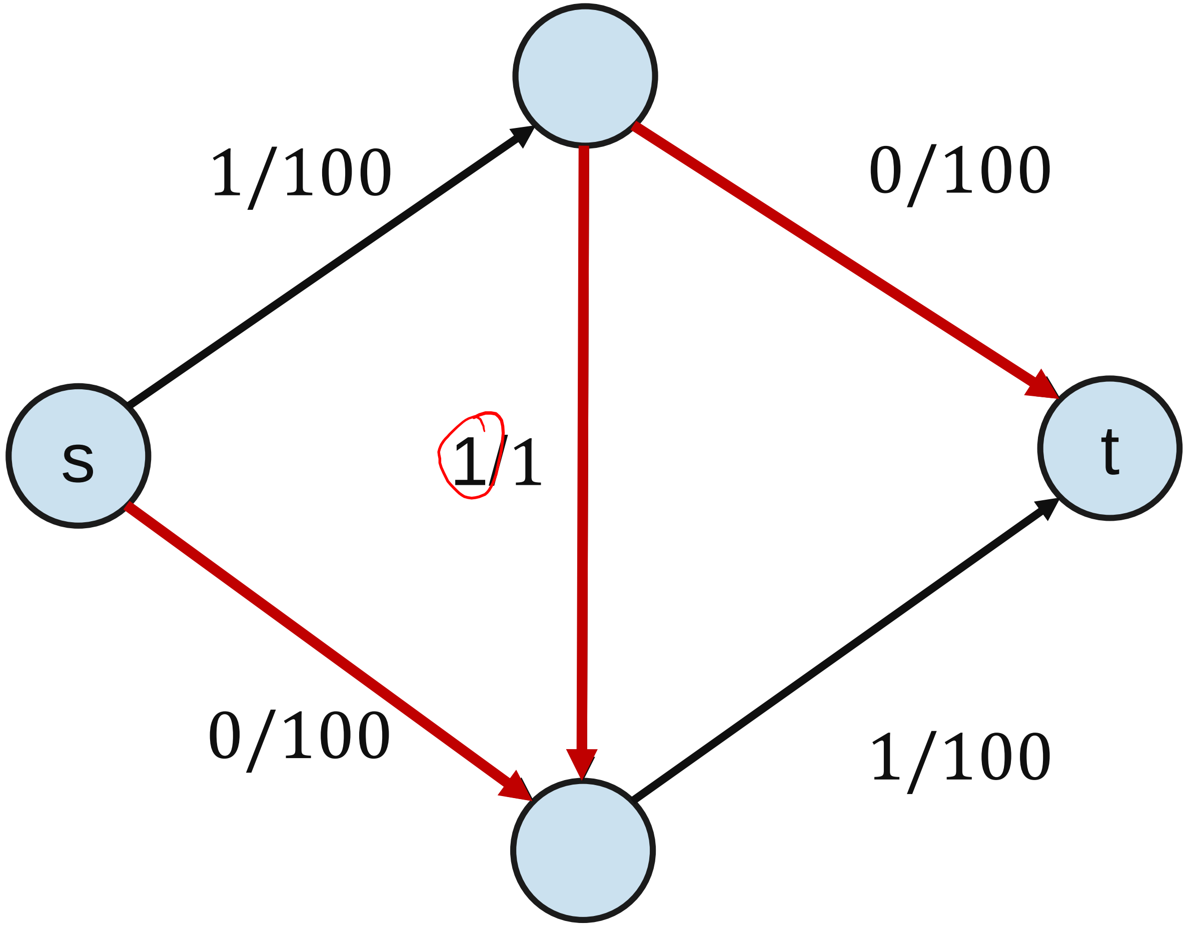
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



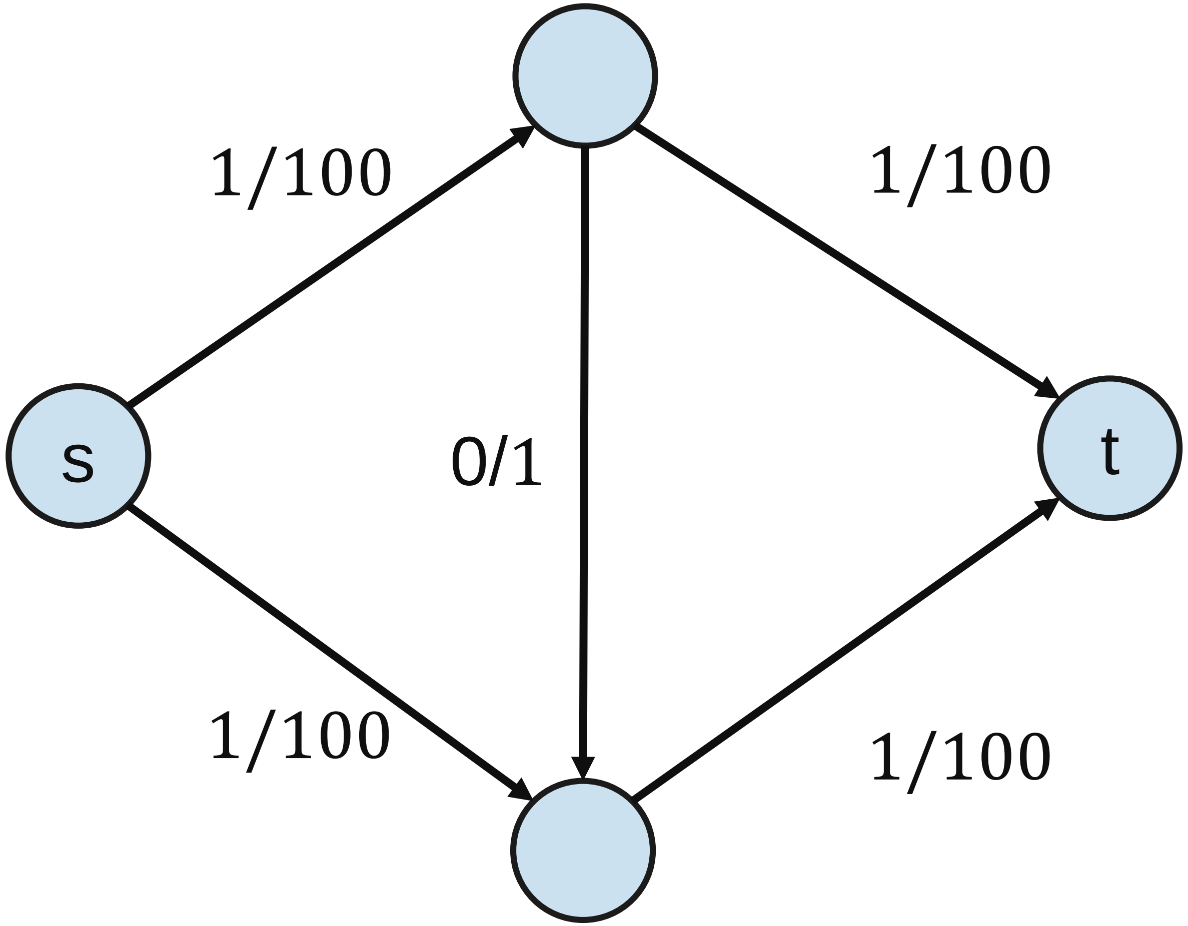
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



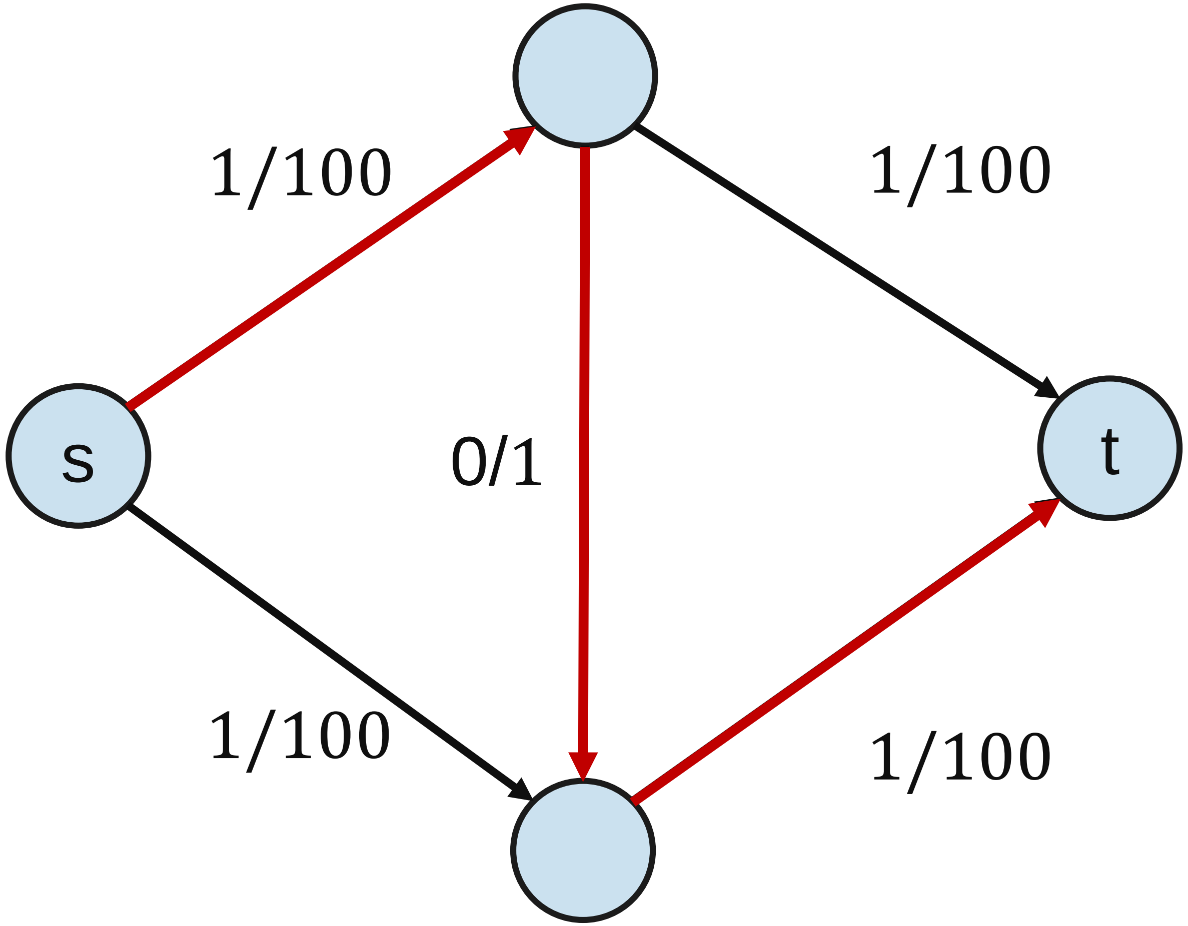
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



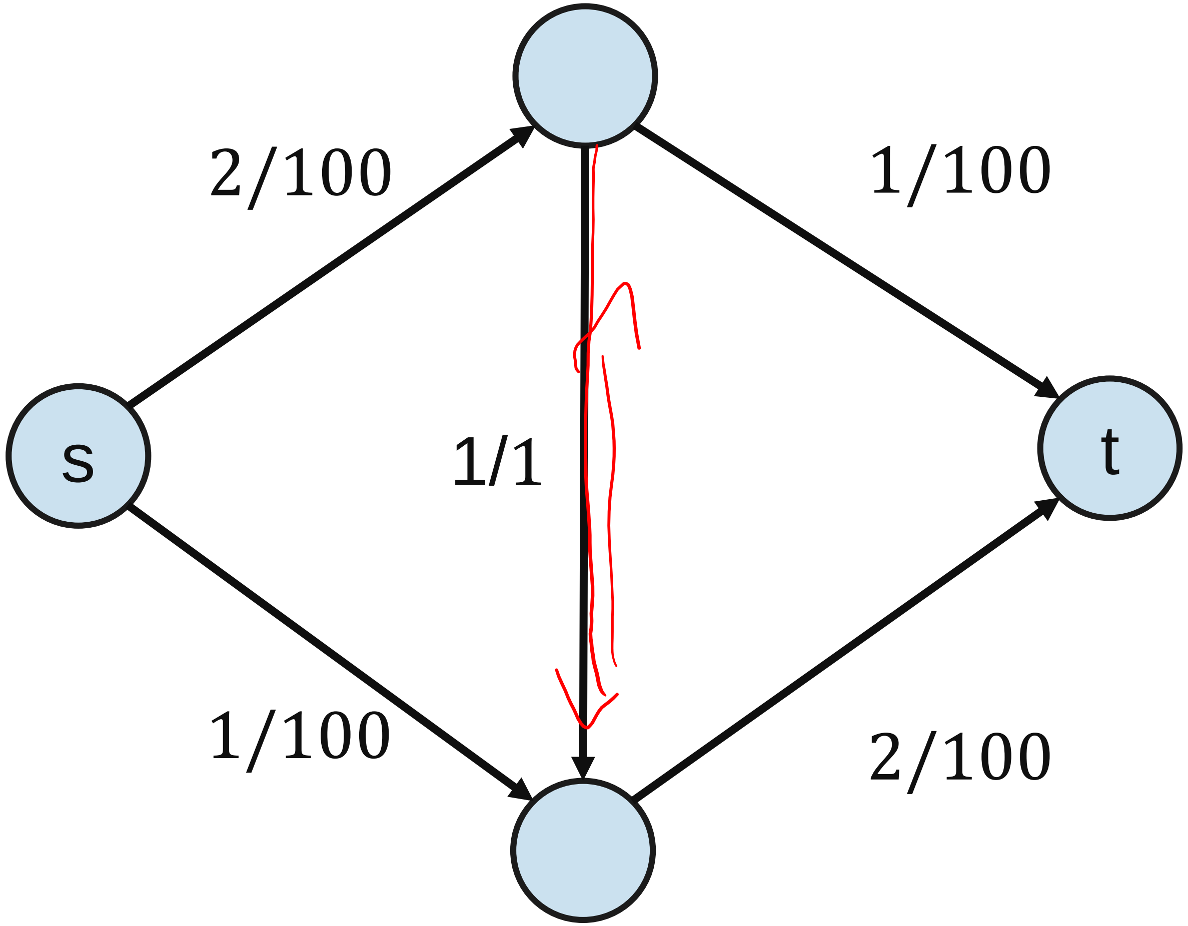
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



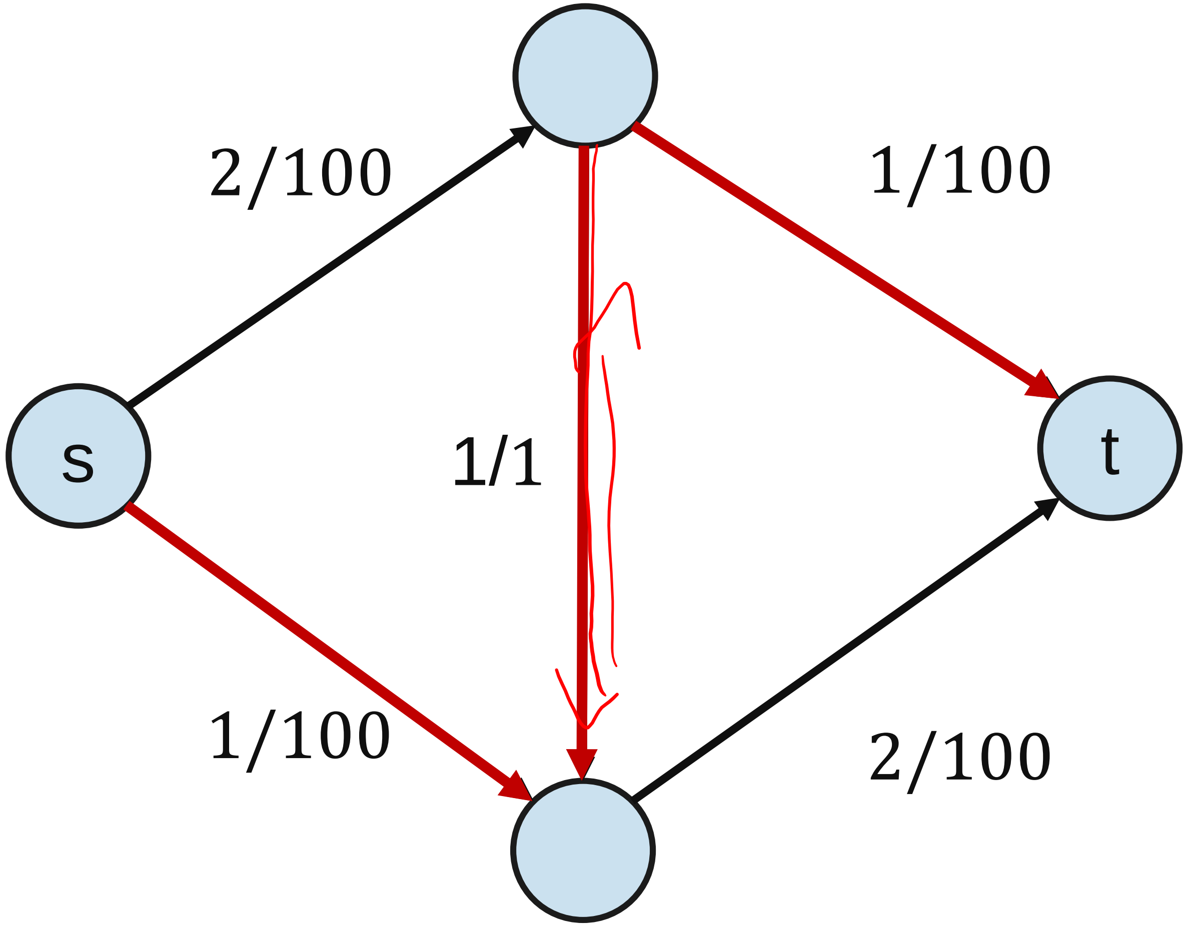
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



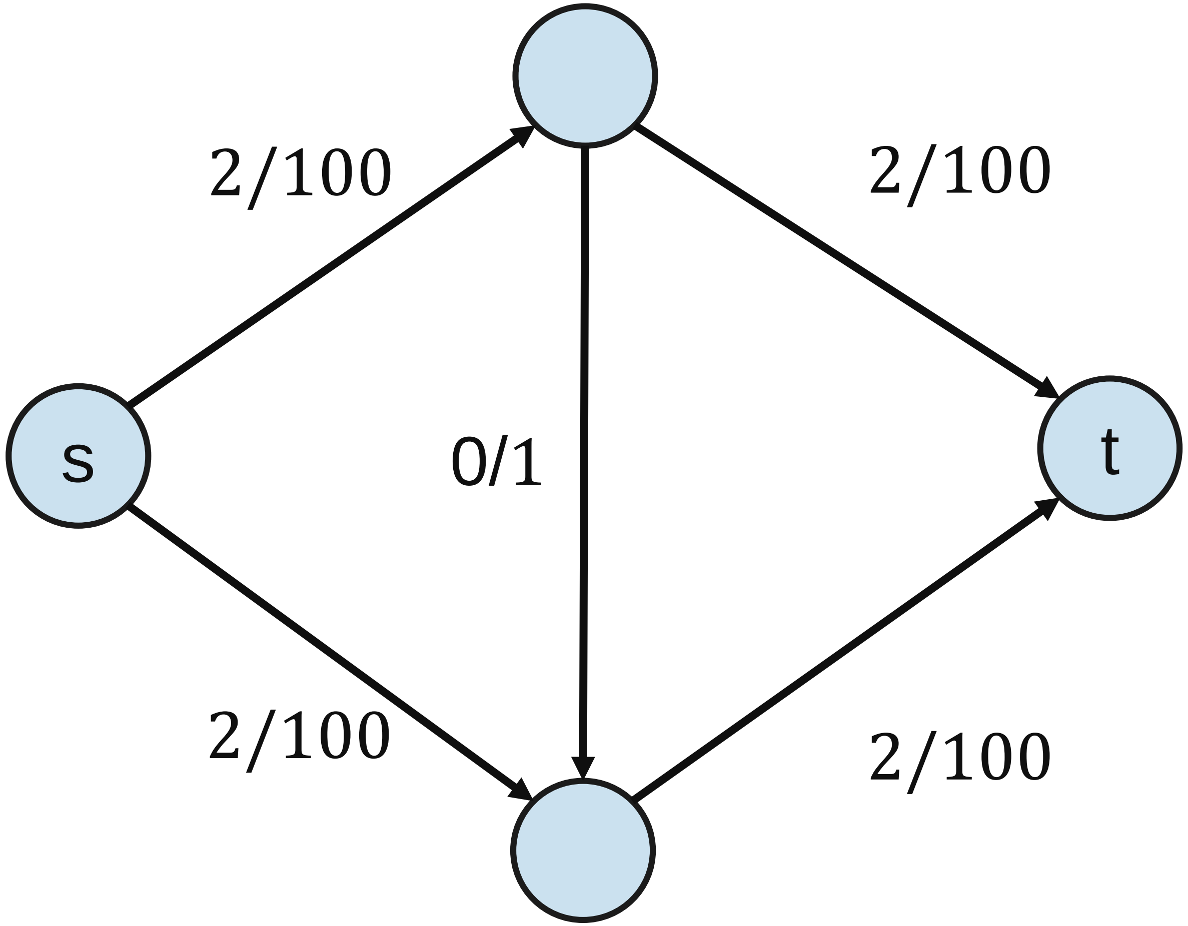
Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



Example: Ford-Fulkerson can be slow

The Ford-Fulkerson-method uses augmentation steps but f^* can be really large …



Might be very slow, even if the network is very small.

Edmonds-Karp-Algorithm

credit also goes to Yefim Dinitz (who actually first published the algorithm/analysis 1970)

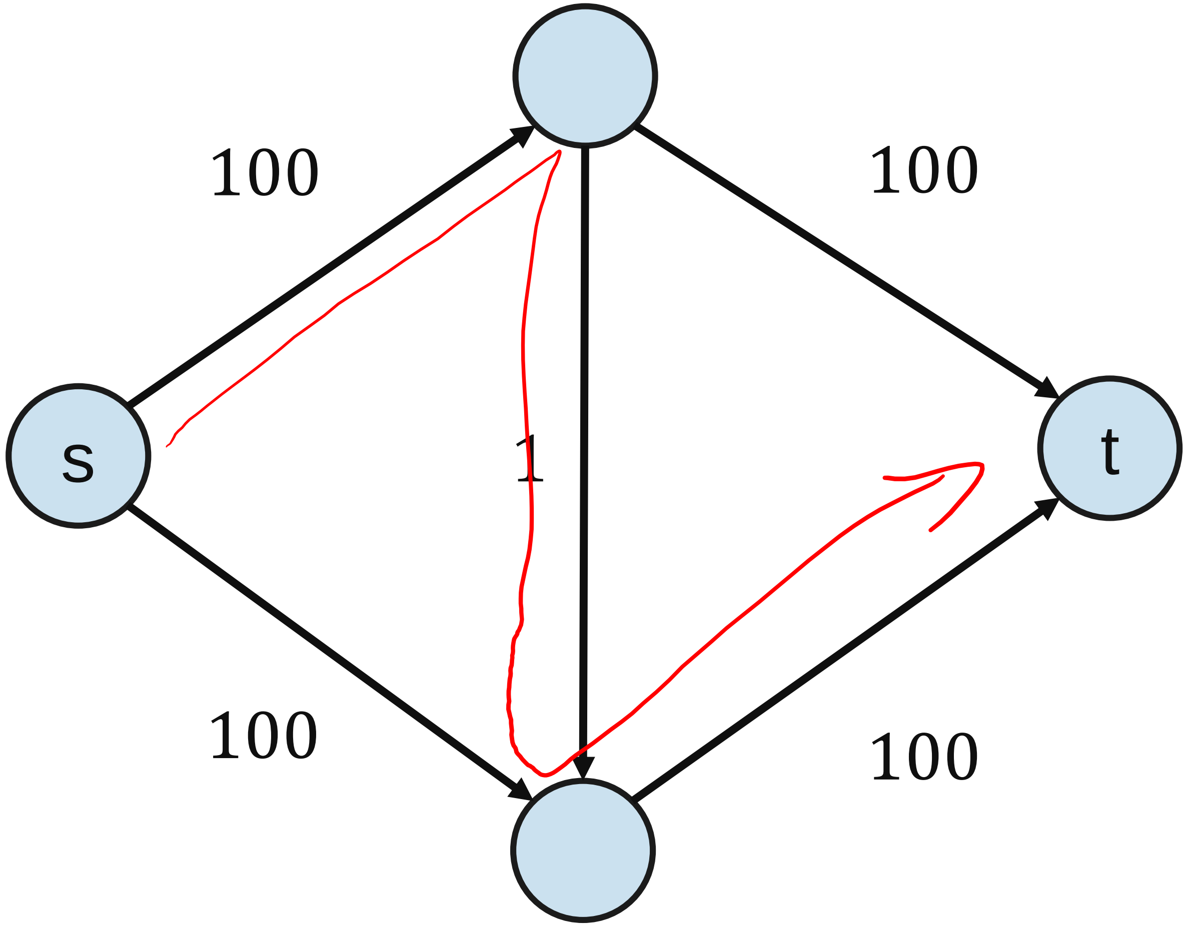
Edmonds-Karp-Algorithm

```
Initialize flow f with 0
while there exists an augmenting path in G_f do
    find a shortest augmenting path p
    augment f along p
return f
```

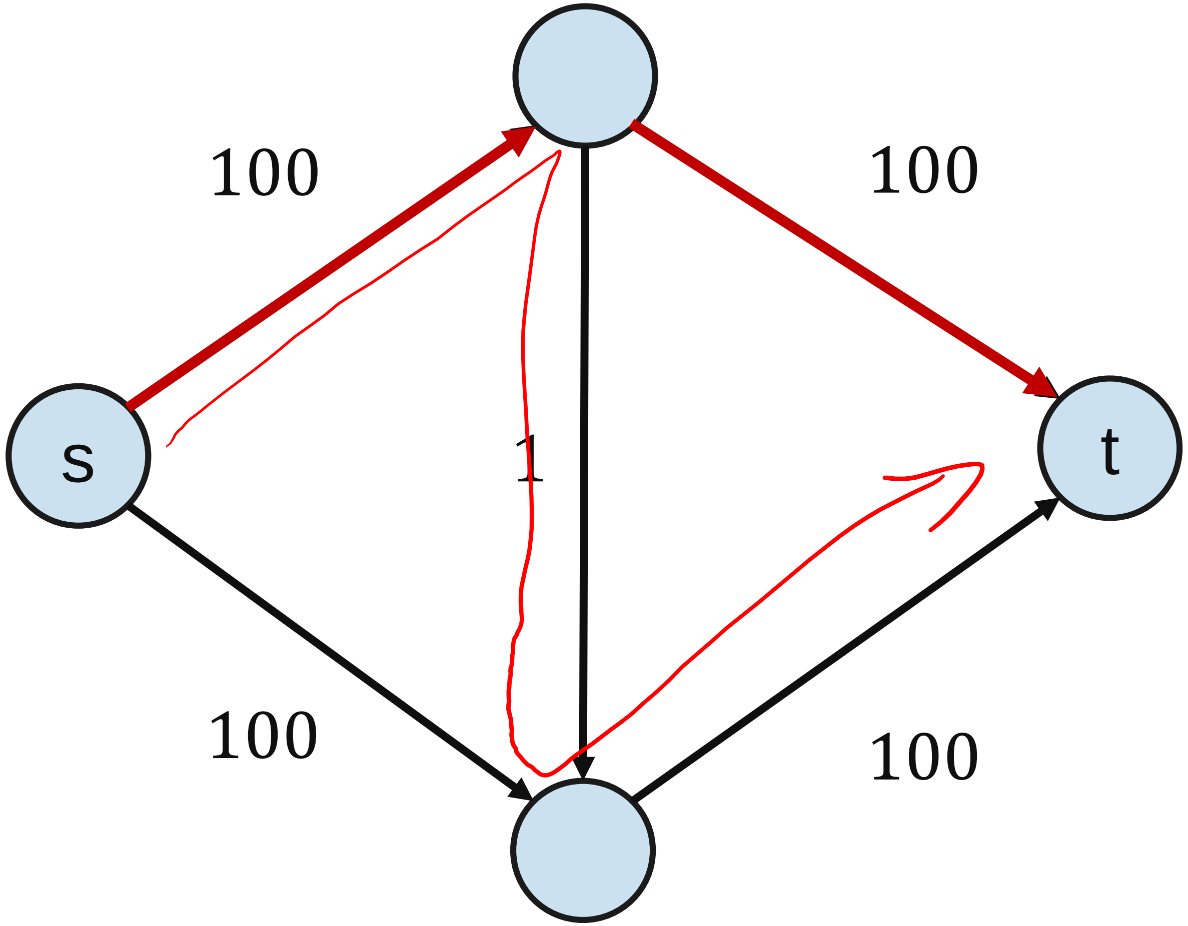
Difference: We always chose a **shortest** augmenting path.

Theorem: The runtime of the Edmonds–Karp–Algorithm is polynomial in the size of the network, regardless of the value of the capacities.

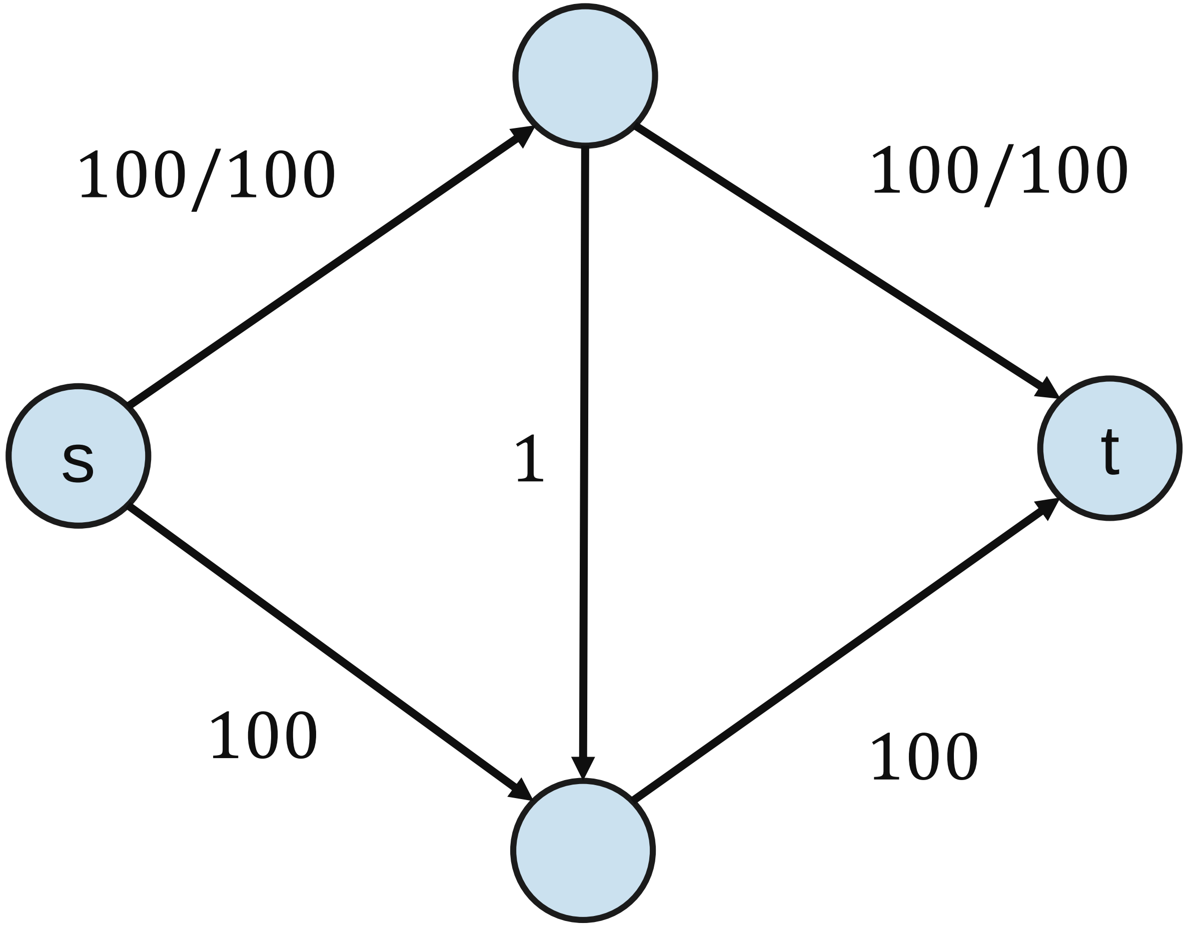
Recap our Example



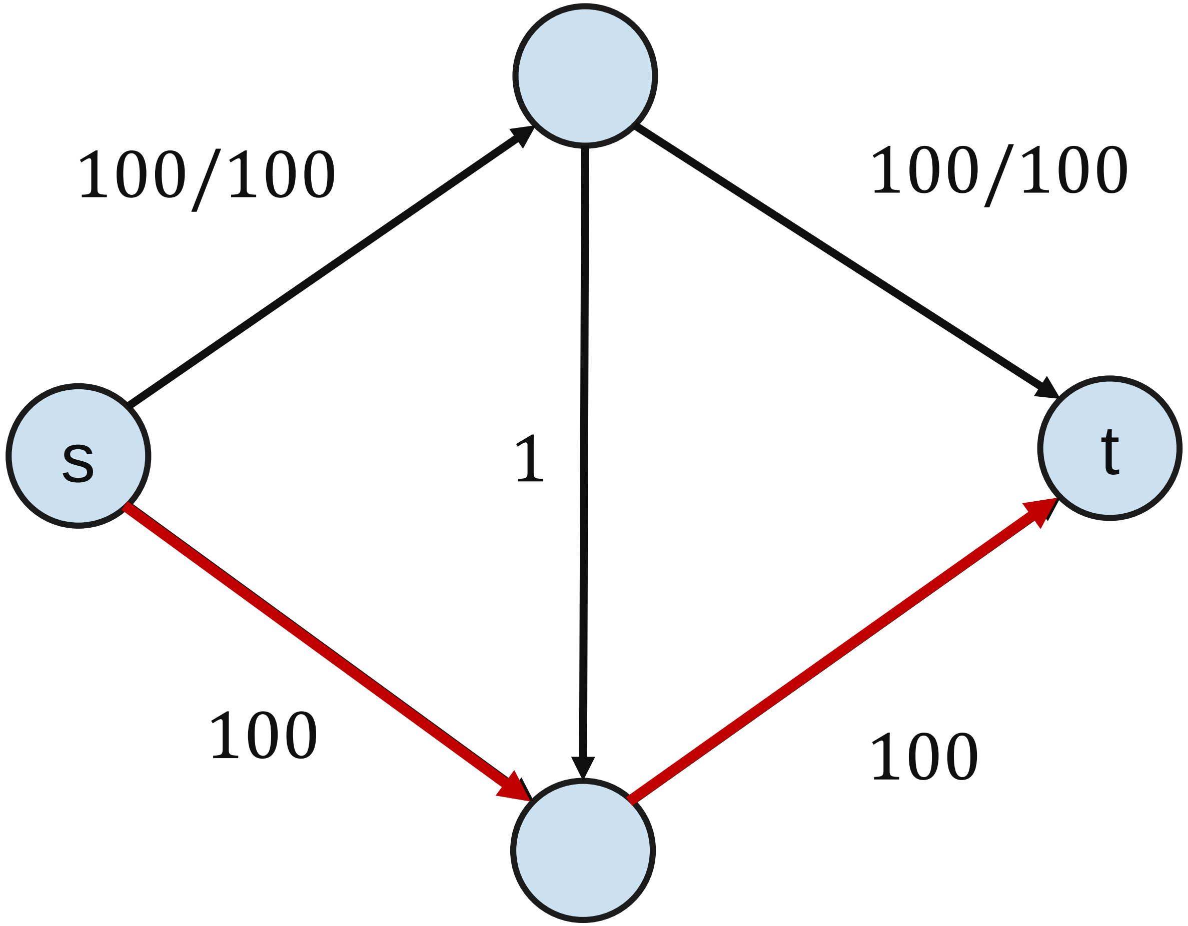
Recap our Example



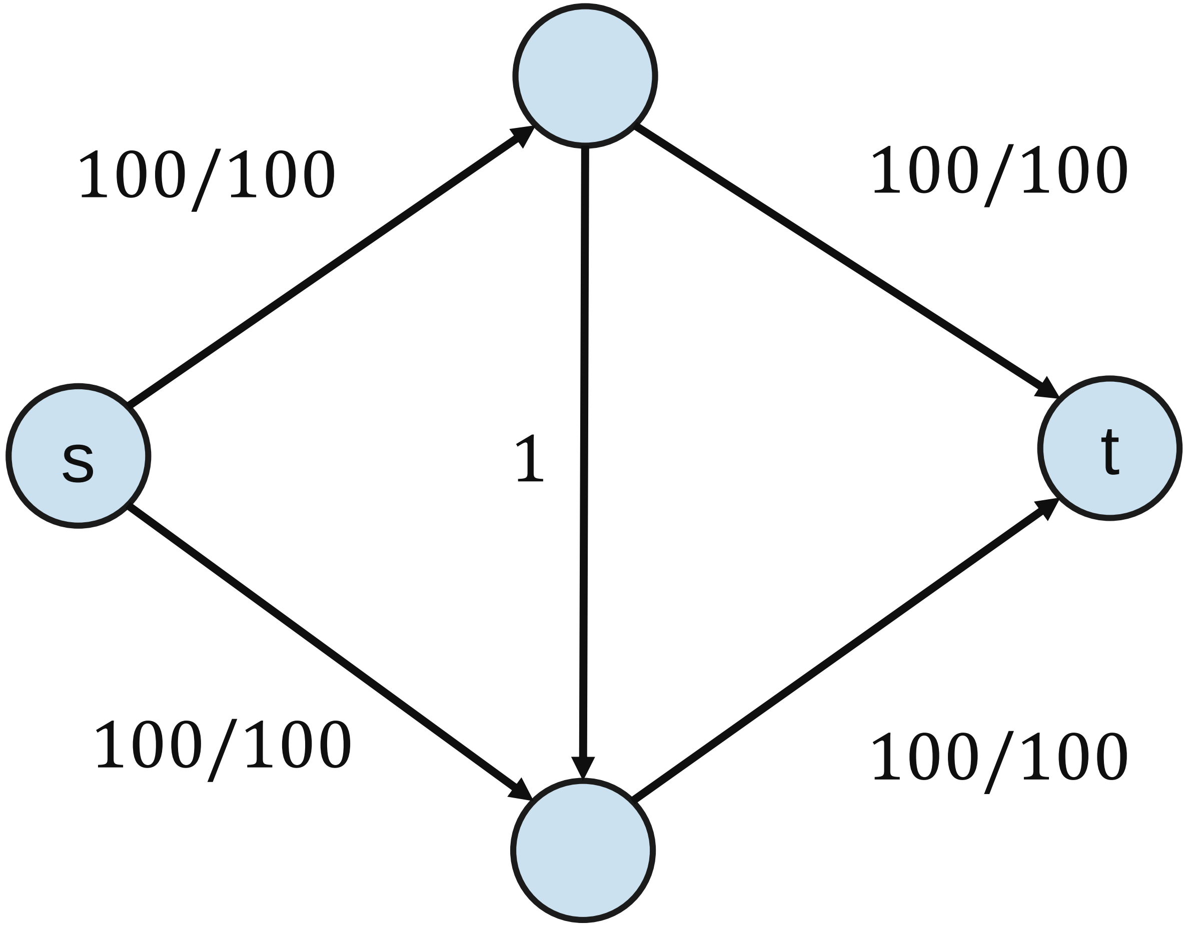
Recap our Example



Recap our Example



Recap our Example



Edmonds-Karp-Algorithm is fast (for this example)!

Analysis of Edmonds-Karp

"We will show that level of a vertex increases if its incident edges appears in several augmentations."

Analysis of Edmonds-Karp

Define:

- f_i: the current flow before the i-th augmentation step.
- $G_i = G_{f_i}$: the residual network of f_i (note that $G_1 = G$)
- $level_i(v)$: the distance of s to v in G_i $\in \{0, 1, \ldots, |V|-1\} \cup \{\infty\}$

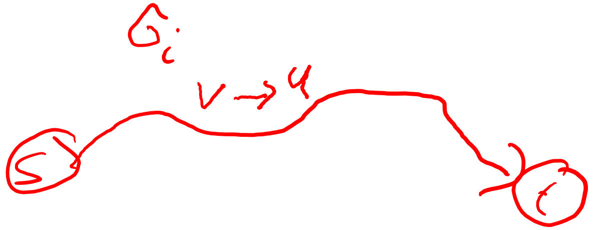
"We will show that level of a vertex increases if its incident edges appears in several augmentations."

Lemma B (levels don't decrease):

For all $v \in V$ and all i we have $level_{i+1}(v) \geq level_i(v)$.

Proof: $level_{i+1}(v) \geq level_i(v)$.

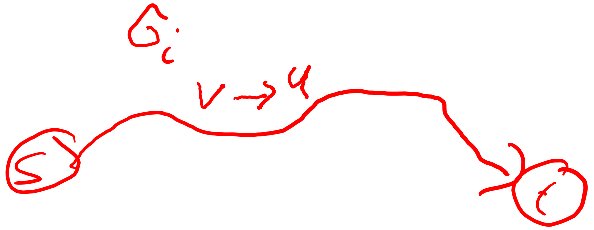
For $v = s$ the claim is trivial as $level_i(s) = 0$.



Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.

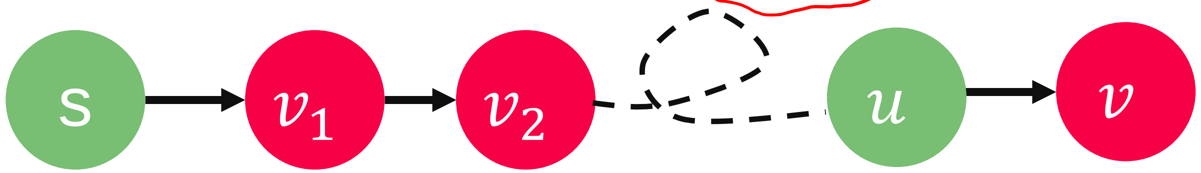


Proof: $level_{i+1}(v) \geq level_i(v)$.

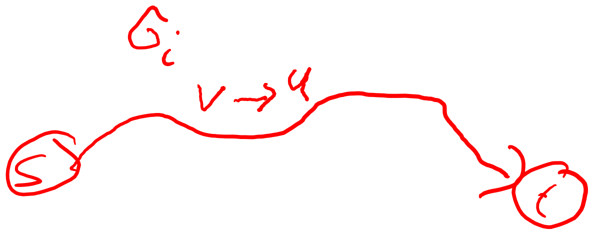
For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

If there is no such path, $level_{i+1}(v) = \infty$, done!

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.



consider a shortest path from s to v in G_{i+1}

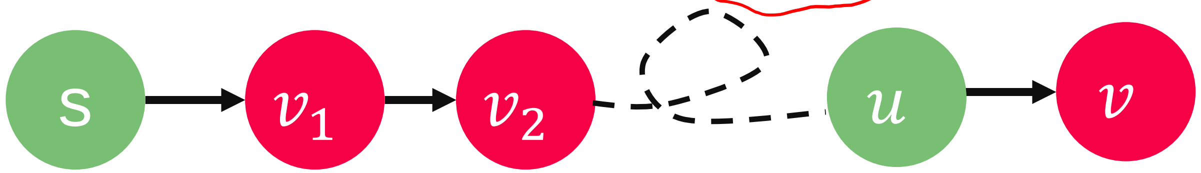


Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

If there is no such path, $level_{i+1}(v) = \infty$, done!

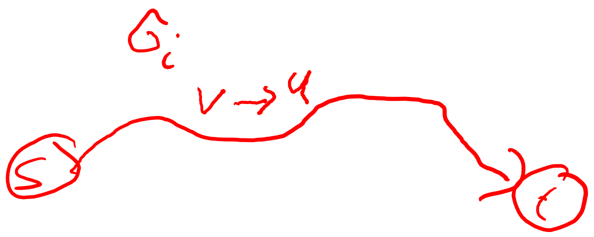
Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.



consider a shortest path from s to v in G_{i+1}

(*) We have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq$ $level_i(u) + 1$

due to choice of v, u doesn't contradict the claim !

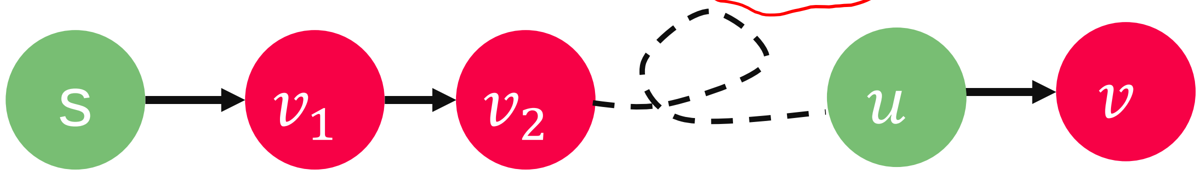


Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\ (s) = 0$.

If there is no such path, $level_{i+1}(v) = \infty$, done!

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.

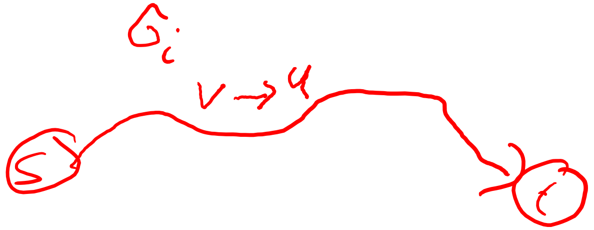


consider a shortest path from s to v in G_{i+1}

(*) We have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq$ $level_i(u) + 1$

due to choice of v, u doesn't contradict the claim !

Case $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{G_i}$: $level_i(v) \leq$ $level_i(u) + 1$
(levels of adjacent vertices can differ by at most one)

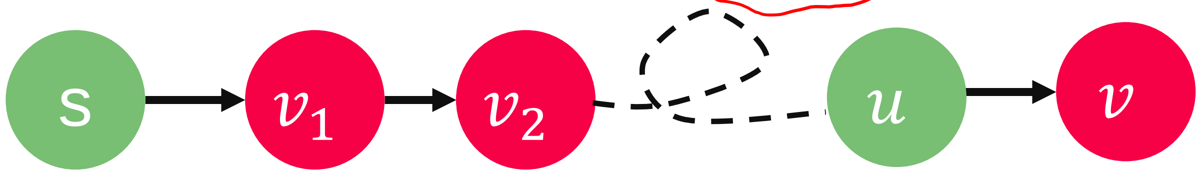


Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

If there is no such path, $level_{i+1}(v) = \infty$, done!

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.

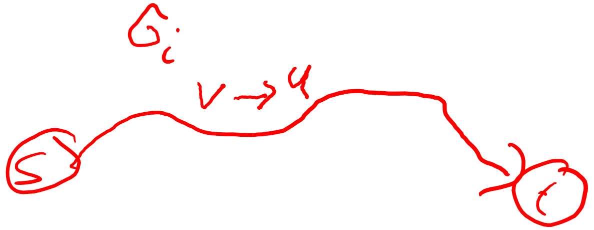


consider a shortest path from s to v in G_{i+1}

(*) We have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq$ $level_i(u) + 1$

due to choice of v, u doesn't contradict the claim !

Case $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{G_i}$: $level_i(v) \leq$ $level_i(u) + 1$ $\overset{(*)}{\leq} level_{i+1}(v)$
(levels of adjacent vertices can differ by at most one)

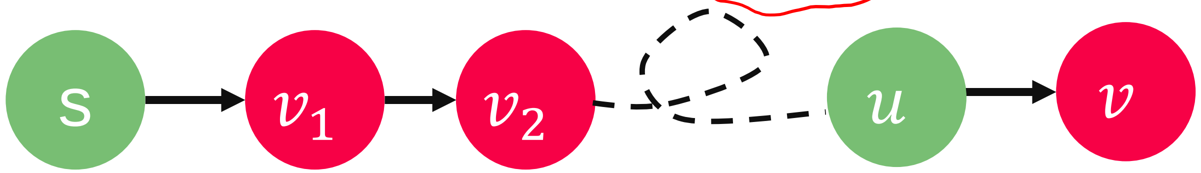


Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

If there is no such path, $level_{i+1}(v) = \infty$, done!

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.



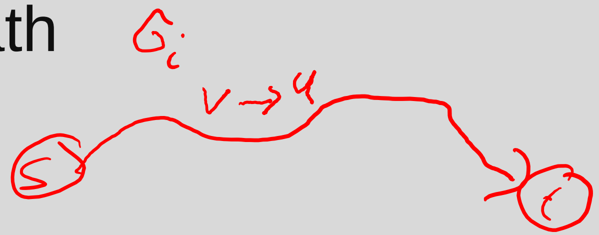
consider a shortest path from s to v in G_{i+1}

(*) We have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq level_i(u) + 1$

due to choice of v, u doesn't contradict the claim !

Case $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{G_i}$: $level_i(v) \leq level_i(u) + 1 \overset{(*)}{\leq} level_{i+1}(v)$
(levels of adjacent vertices can differ by at most one)

Case $(\boldsymbol{u}, \boldsymbol{v}) \notin \boldsymbol{G_i}$: (v, u) must be an edge in the i-th augmenting path
(v, u) must lie on the shortest s-t-path in G_i as (u, v) in G_{i+1}, hence
$level_i(v) = level_i(u) - 1 \underset{\text{trivial}}{\leq} level_i(u) + 1$

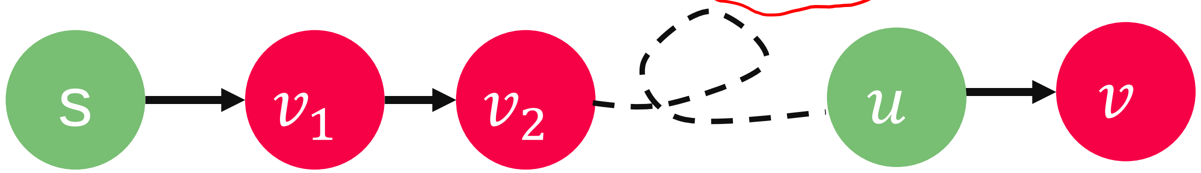


Proof: $level_{i+1}(v) \geq level_i(v)$.

For $v = s$ the claim is trivial as $level_i\,(s) = 0$.

Let $v \neq s$ with minimal $level_{i+1}(v)$, contradicting the claim.

If there is no such path, $level_{i+1}(v) = \infty$, done!



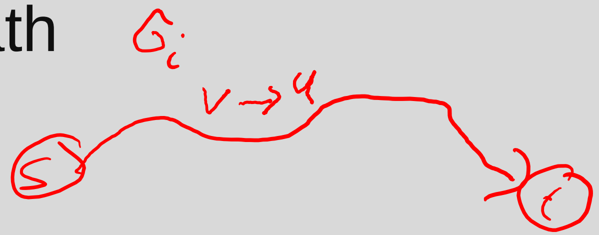
consider a shortest path from s to v in G_{i+1}

(*) We have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq$ $level_i(u) + 1$

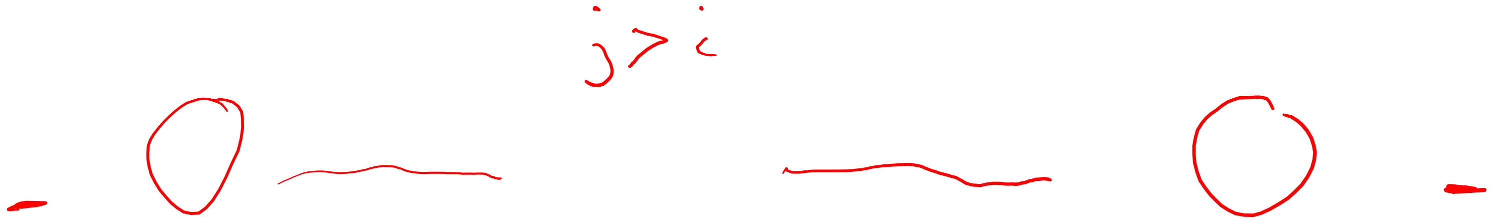
due to choice of v, u doesn't contradict the claim !

Case $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{G_i}$: $level_i(v) \leq level_i(u) + 1 \overset{(*)}{\leq} level_{i+1}(v)$
(levels of adjacent vertices can differ by at most one)

Case $(\boldsymbol{u}, \boldsymbol{v}) \notin \boldsymbol{G_i}$: (v, u) must be an edge in the i-th augmenting path
(v, u) must lie on the shortest s-t-path in G_i as (u, v) in G_{i+1}, hence
$level_i(v) = level_i(u) - 1 \underset{\text{trivial}}{\leq} level_i(u) + 1 \underset{(*)}{\leq} level_{i+1}(v)$



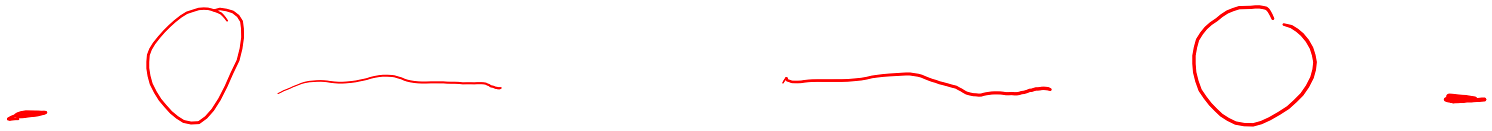
During the execution of the Edmonds-Karp-Algorithm, any edge (u, v) disappears from the residual graph at most $V/2$ times.



During the execution of the Edmonds-Karp-Algorithm, any edge (u, v) disappears from the residual graph at most $V/2$ times.

Proof: Suppose $u \rightarrow v$ is in two residual graphs G_i and G_{j+1}, but not in any of the intermediate residual graphs G_{i+1}, G_j for some i < j.

$j > i$



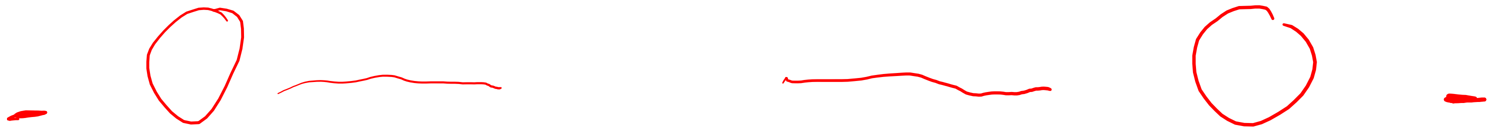
Edmonds-Karp-Algorithm: Runtime Analysis

During the execution of the Edmonds-Karp-Algorithm, any edge (u, v) disappears from the residual graph at most $V/2$ times.

Proof: Suppose $u \rightarrow v$ is in two residual graphs G_i and G_{j+1}, but not in any of the intermediate residual graphs G_{i+1}, G_j for some i < j.

- $u \rightarrow v$ must be on the i-th augmenting path: $level_i(v) = level_i(u) + 1$,
- $v \rightarrow u$ must be on the j-th augmenting path: $level_j(v) = level_j(u) - 1$.

$j > i$



Edmonds-Karp-Algorithm: Runtime Analysis

During the execution of the Edmonds-Karp-Algorithm, any edge (u, v) disappears from the residual graph at most $V/2$ times.

Proof: Suppose $u \rightarrow\ v$ is in two residual graphs G_i and G_{j+1}, but not in any of the intermediate residual graphs G_{i+1}, G_j for some i < j.

- $u \rightarrow\ v$ must be on the i-th augmenting path: $level_i(v)\ = level_i(u)\ +\ 1,$
- $v \rightarrow u$ must be on the j-th augmenting path: $level_j(v)\ =\ level_j(u)\ -\ 1.$

By Lemma B, we have

$j > i$

$$level_j(u) = level_j(v) +\ 1\ \geq\ level_i(v) +\ 1 = level_i\ (u)\ +\ 2.$$

During the execution of the Edmonds-Karp-Algorithm, any edge (u, v) disappears from the residual graph at most $V/2$ times.

Proof: Suppose $u \to v$ is in two residual graphs G_i and G_{j+1}, but not in any of the intermediate residual graphs G_{i+1}, G_j for some i < j.

- $u \to v$ must be on the i-th augmenting path: $level_i(v) = level_i(u) + 1$,
- $v \to u$ must be on the j-th augmenting path: $level_j(v) = level_j(u) - 1$.

By Lemma B, we have

$j > i$

$$level_j(u) = level_j(v) + 1 \geq level_i(v) + 1 = level_i(u) + 2.$$

The distance from s to u increased by at least 2 between the disappearance and reappearance of $u \to v$. Since every level is either less than V or infinite, the number of disappearances is at most $V/2$

Edmonds-Karp-Algorithm

The Edmonds-Karp-Algorithm computes a maximum flow with runtime $O(|E|^2|V|)$.

Edmonds-Karp-Algorithm

The Edmonds-Karp-Algorithm computes a maximum flow with runtime $O(|E|^2|V|)$.

Proof:

- As a special case of the Ford-Fulkerson Method, it computes a max-flow!

Edmonds-Karp-Algorithm

The Edmonds-Karp-Algorithm computes a maximum flow with runtime $O(|E|^2|V|)$.

Proof:

- As a special case of the Ford-Fulkerson Method, it computes a max-flow!
- Every augmentation requires $O(|E|)$ steps as a shortest augmenting paths can be found via BFS and also G_f can be built with $O(|E|)$ steps.

Edmonds-Karp-Algorithm

The Edmonds-Karp-Algorithm computes a maximum flow with runtime $O(|E|^2|V|)$.

Proof:

- As a special case of the Ford-Fulkerson Method, it computes a max-flow!
- Every augmentation requires $O(|E|)$ steps as a shortest augmenting paths can be found via BFS and also G_f can be built with $O(|E|)$ steps.
- There are at most $O(|E| \cdot |V|)$ augmentations ($\leq |V|/2$ augmentations per edge)

State-of-The-Art

Ford-Fulkerson Naïve, 1956	$O(Ef^*)$
Edmonds-Karp/Dinic's Algorithm, 1970	$O(E^2V)$ $O(V^5)$
Dinic's Algorithm, 1970	$O(EV^2) = O(V^4)$

State-of-The-Art

Ford-Fulkerson Naïve, 1956	$O(Ef^*)$
Edmonds-Karp/Dinic's Algorithm, 1970	$O(E^2V)$ $O(V^5)$
Dinic's Algorithm, 1970	$O(EV^2) = O(V^4)$

MKM (Malhotra, Kumar, Maheshwari), 1978	$O(V^3)$
Dinic's algorithm with dynamic trees, 1982	$O(VE\log V)$
Push-relabel (many versions exist), 1988	$O(V^2E)$
KRT (King, Rao, Tarjan)'s algorithm, 1994	$O(VE\log_{\frac{E}{V\log V}} V)$

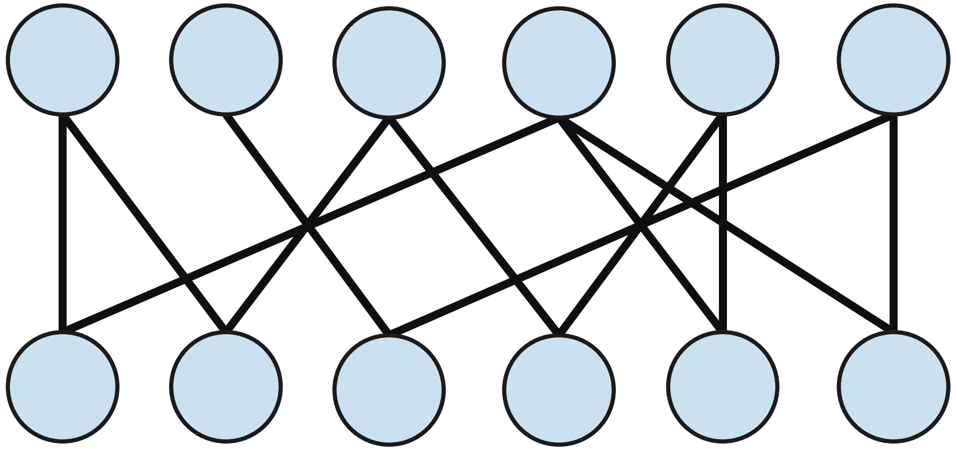
State-of-The-Art

Ford-Fulkerson Naïve, 1956	$O(Ef^*)$
Edmonds-Karp/Dinic's Algorithm, 1970	$O(E^2V)$ $O(V^5)$
Dinic's Algorithm, 1970	$O(EV^2) = O(V^4)$

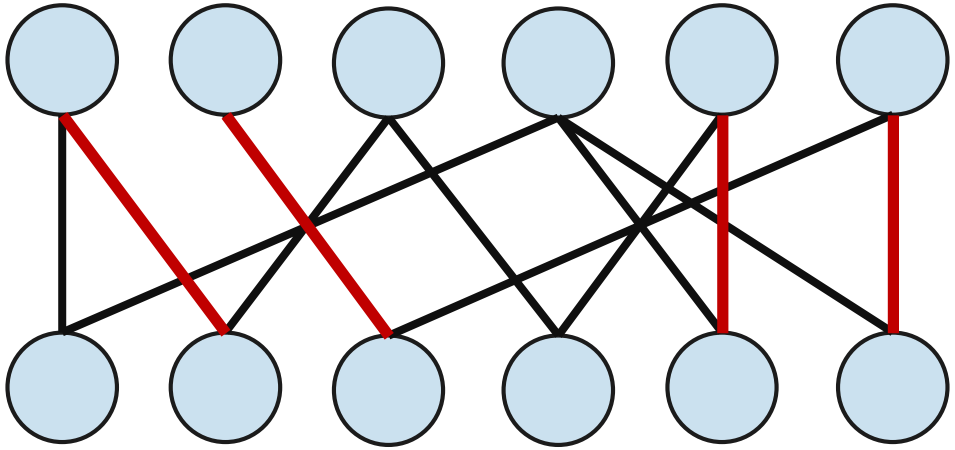
MKM (Malhotra, Kumar, Maheshwari), 1978	$O(V^3)$
Dinic's algorithm with dynamic trees, 1982	$O(VE \log V)$
Push-relabel (many versions exist), 1988	$O(V^2E)$
KRT (King, Rao, Tarjan)'s algorithm, 1994	$O(VE \log_{\frac{E}{V \log V}} V)$
…	
Orlin + KRT, 2013	$O(VE)$
Gao-Liu-Peng, 2021	$\tilde{O}(E^{\frac{3}{2}-\frac{1}{328}} \log maxCapacity)$

Complicated, rely on involved Data structures

Application: Maximum Cardinality Bipartite Matching



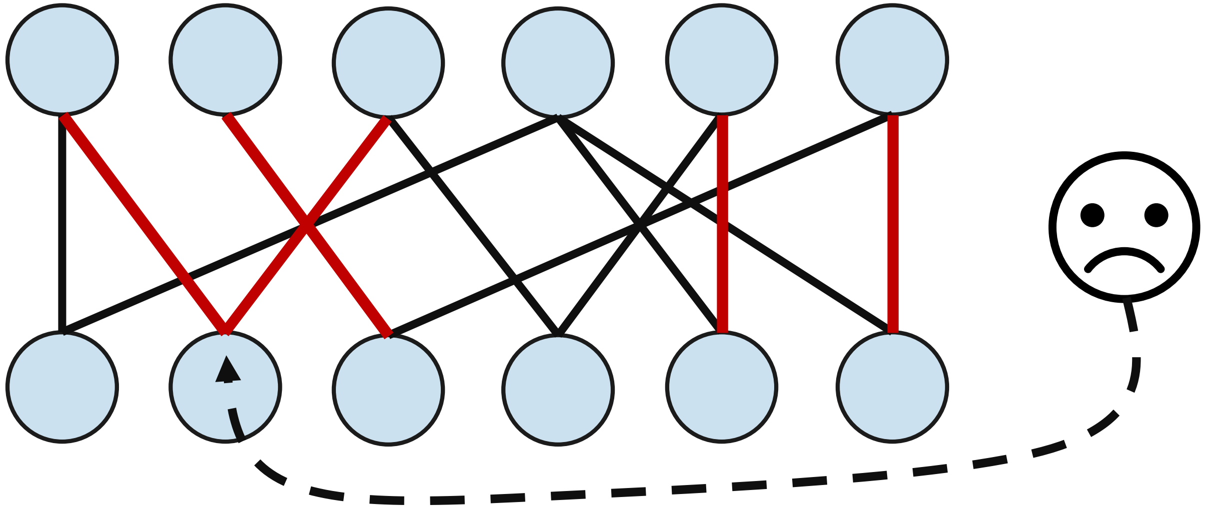
A matching of a graph $G = (V, E)$ is a subset $M \subseteq E$ of the edges such any vertex $v \in V$ has at most one adjacent edge in M.



☺

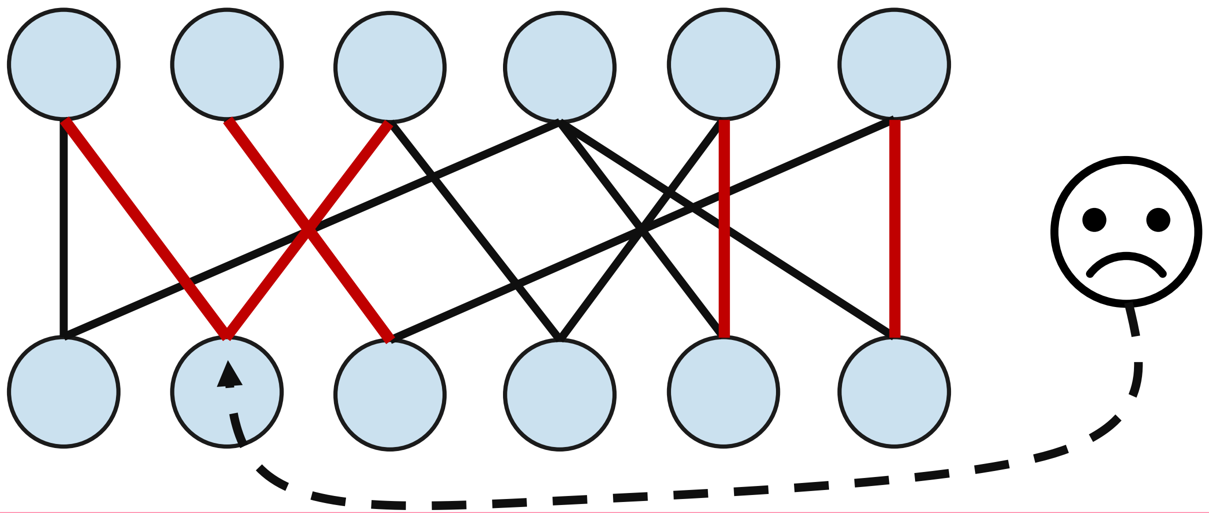
Application: Maximum Bipartite Cardinality Matching

A matching of a graph $G = (V, E)$ is a subset $M \subseteq E$ of the edges such any vertex $v \in V$ has at most one adjacent edge in M.



Application: Maximum Bipartite Cardinality Matching

A matching of a graph $G = (V, E)$ is a subset $M \subseteq E$ of the edges such any vertex $v \in V$ has at most one adjacent edge in M.

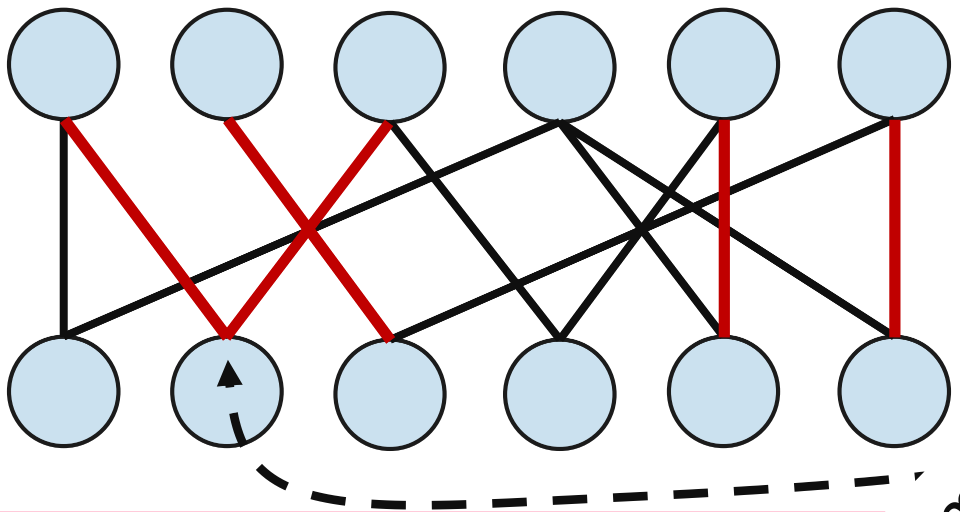


Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.

Here, important special case: The graph is bipartite!

(match customers to products, ads to customers, …)

A matching of a graph $G = (V, E)$ is a subset $M \subseteq E$ of the edges such any vertex $v \in V$ has at most one adjacent edge in M.



Many extensions exist: minimum cost bipartite matchings, …

Today: The simplest version

Maximum Bipartite Cardinality Matching: G… bipartite graph, compute a matching that contains as m… es as possible.

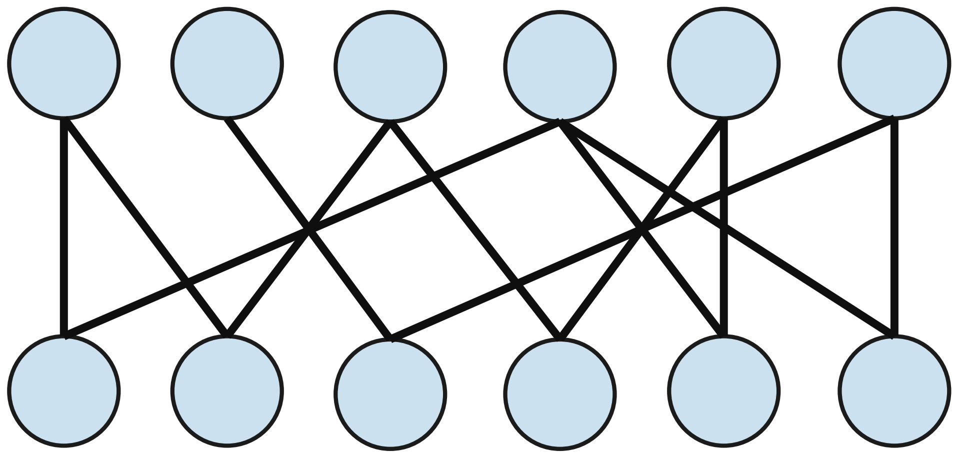
Here, important special case: T… n is bipartite!

(match customers to product… o customers, …)

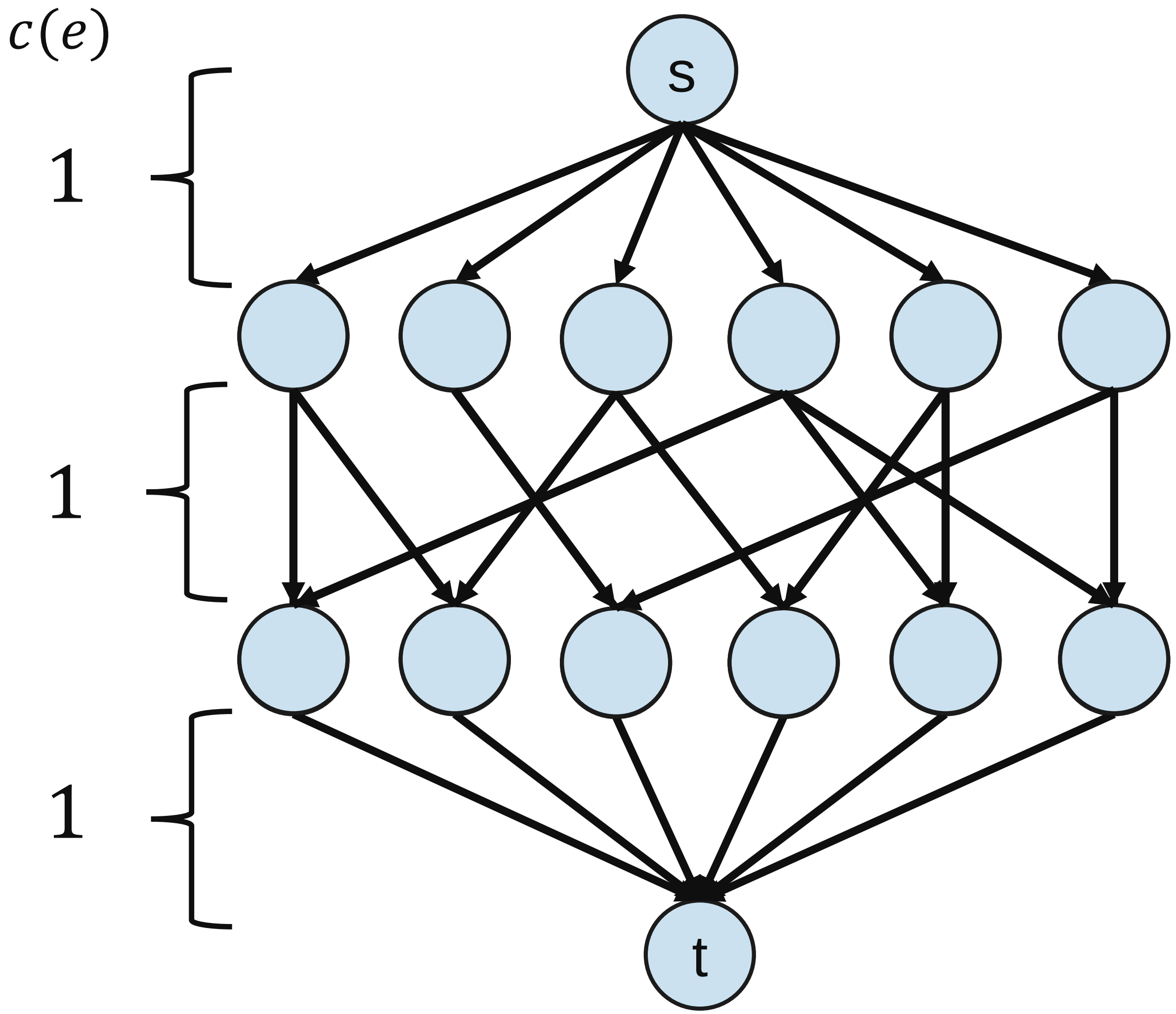
Application: Maximum Bipartite Cardinality Matching

Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.

Reduction to Max-Flow



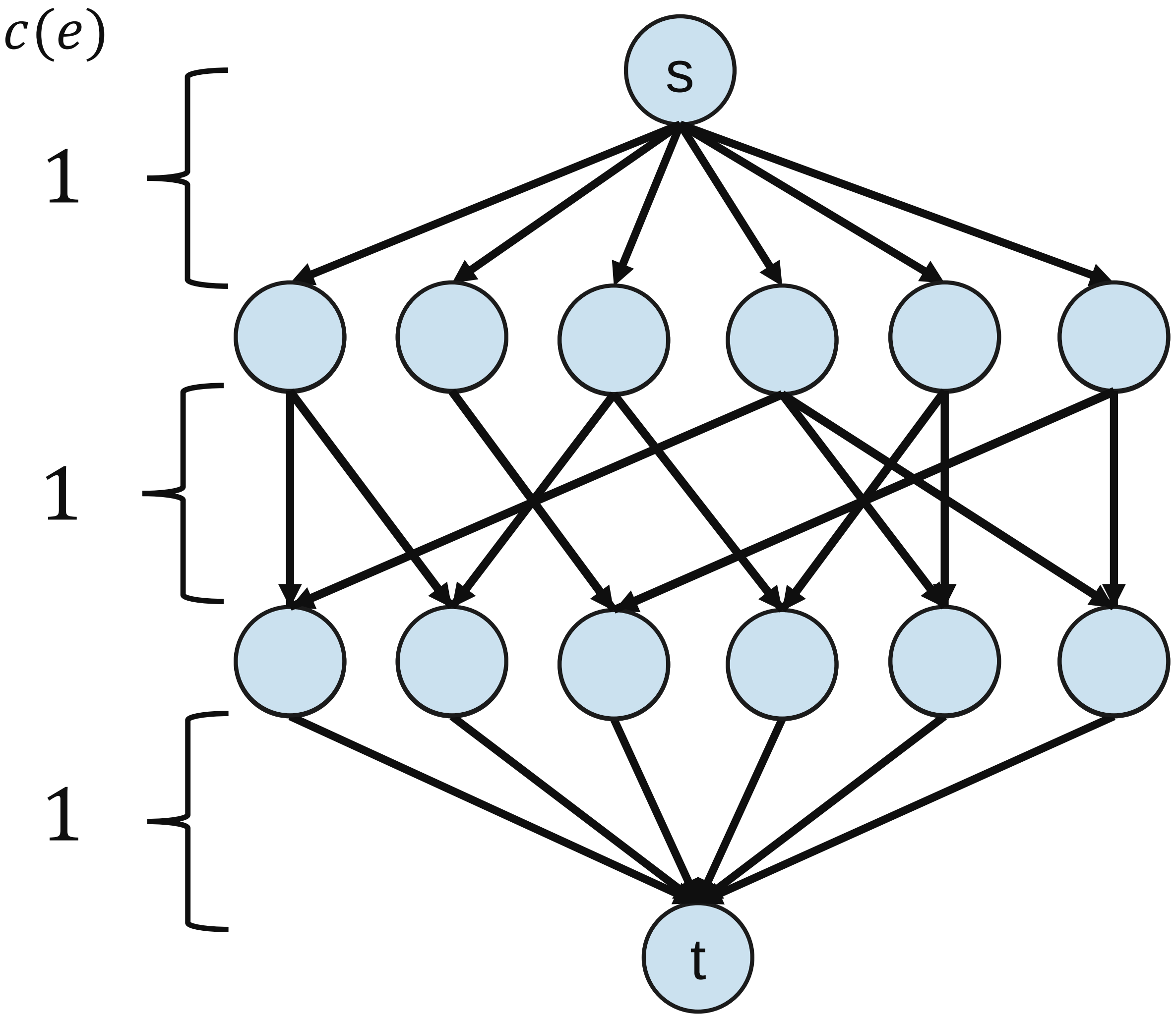
Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.



Reduction to Max-Flow

Application: Maximum Bipartite Cardinality Matching

Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.



Reduction to Max-Flow

Given: Bipartite graph $B = (X \cup Y, E)$.

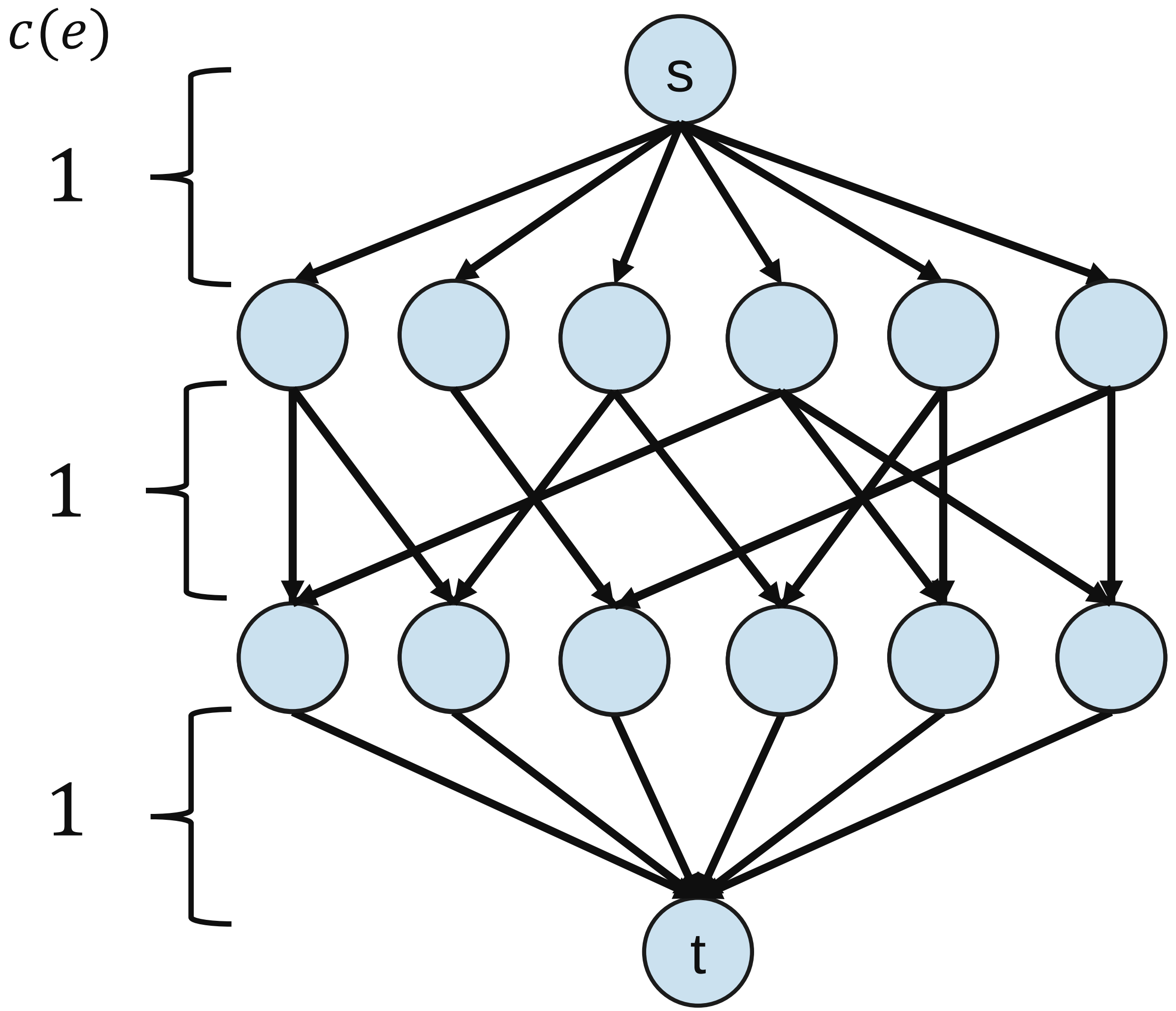
1. Build flow network

$$\tilde{B} = (X \cup Y \cup \{s,t\}, E', c) \text{ with}$$

$$E' = \{(s,x) | x \in X\} \cup \{(y,t) | y \in Y\} \cup \{(x,y) | \{x,y\} \in E\}$$

and $c(e') = 1$ for all $e' \in E'$.

Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.



Reduction to Max-Flow

Given: Bipartite graph $B = (X \cup Y, E)$.

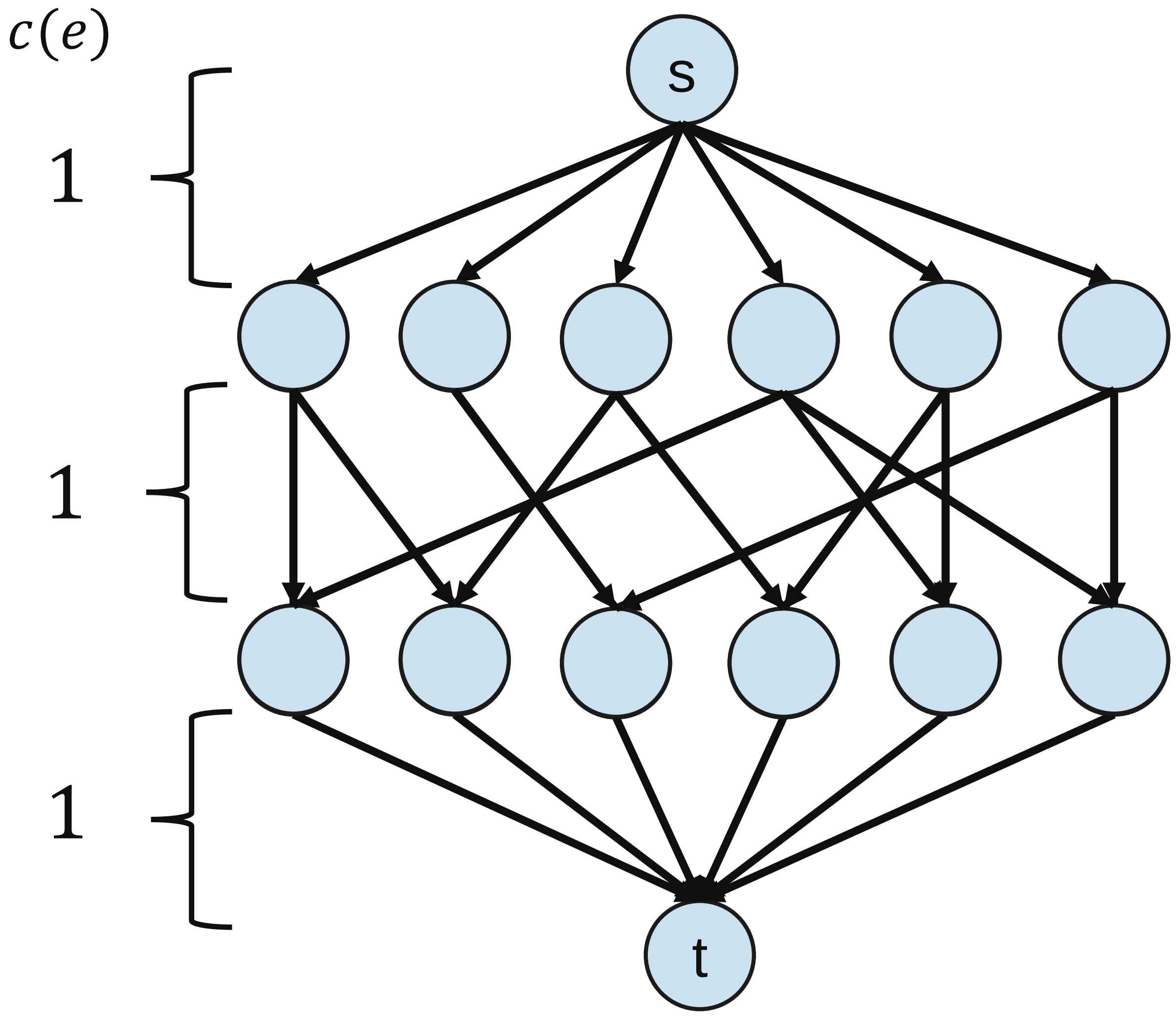
1. Build flow network

   $\tilde{B} = (X \cup Y \cup \{s,t\}, E', c)$ with
   $E' = \{(s,x) | x \in X\} \cup \{(y,t) | y \in Y\}$
   $\cup \{(x,y) | \{x,y\} \in E\}$
   and $c(e') = 1$ for all $e' \in E'$.

2. Compute an **integral** max-flow of $(\tilde{B}, c)$
3. Return the edges (of the original graph) with flow 1

Application: Maximum Bipartite Cardinality Matching

Maximum Bipartite Cardinality Matching: Given a bipartite graph, compute a matching that contains as many edges as possible.



Reduction to Max-Flow

Given: Bipartite graph $B = (X \cup Y, E)$.

1. Build flow network

   $(\widetilde{B} = (X \cup Y \cup \{s,t\}, E', c)$ with

   $E' = \{(s,x) | x \in X\} \cup \{(y,t) | y \in Y\}$

   $\cup \{(x,y) | \{x,y\} \in E\}$

   and $c(e') = 1$ for all $e' \in E'$.

2. Compute an **integral** max-flow of $(\tilde{B}, c)$
3. Return the edges (of the original graph) with flow 1

Proof: Omitted, see, e.g., CLRS.

What else?

Link to animated version: https://algorithms.discrete.ma.tum.de/

Thank you