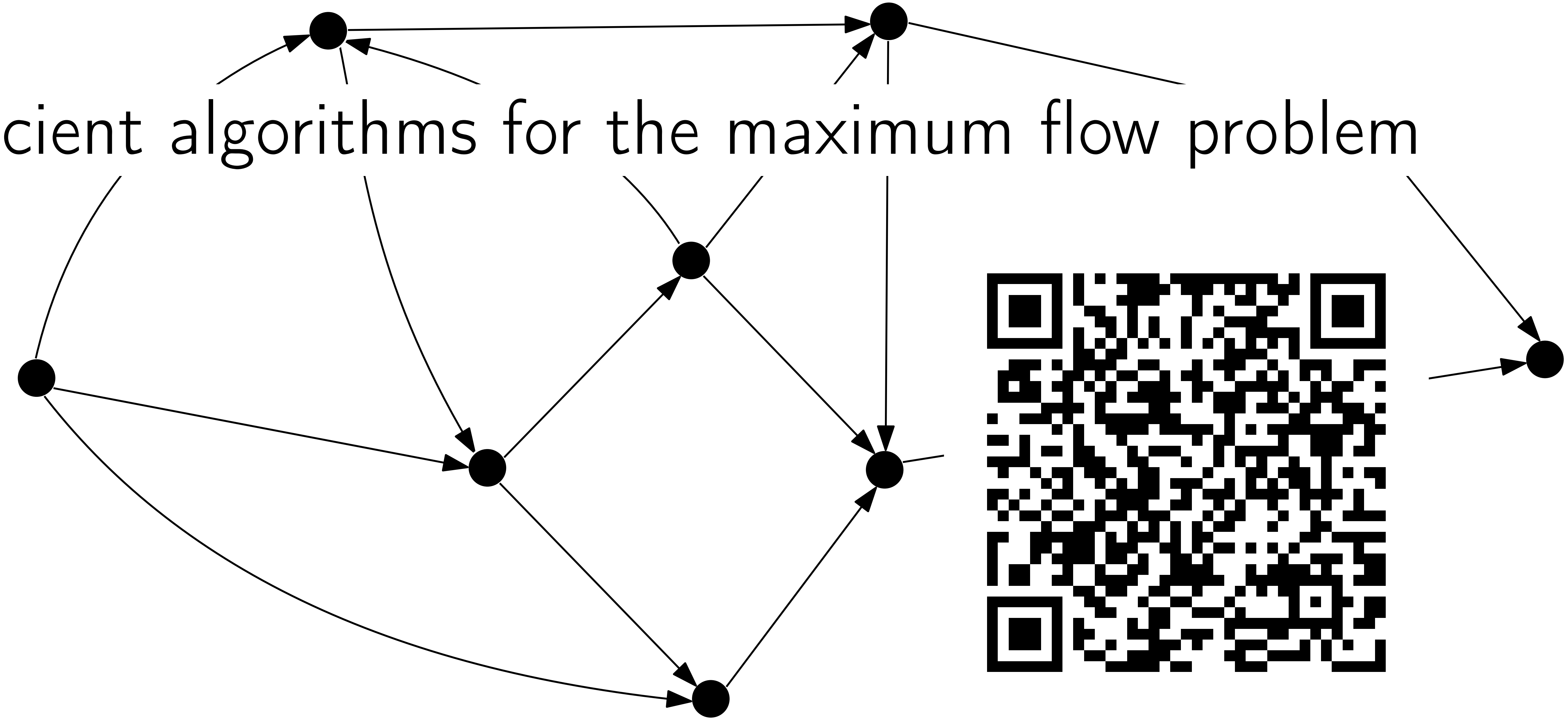
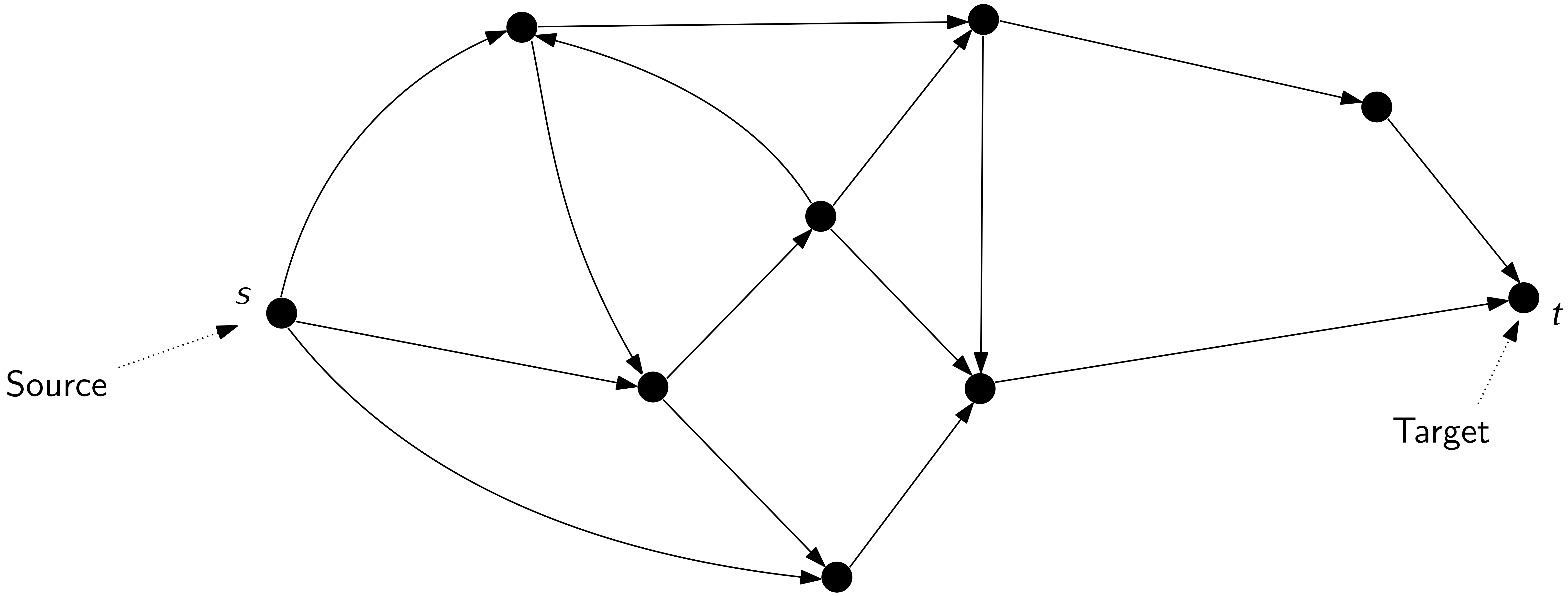
Efficient algorithms for the maximum flow problem



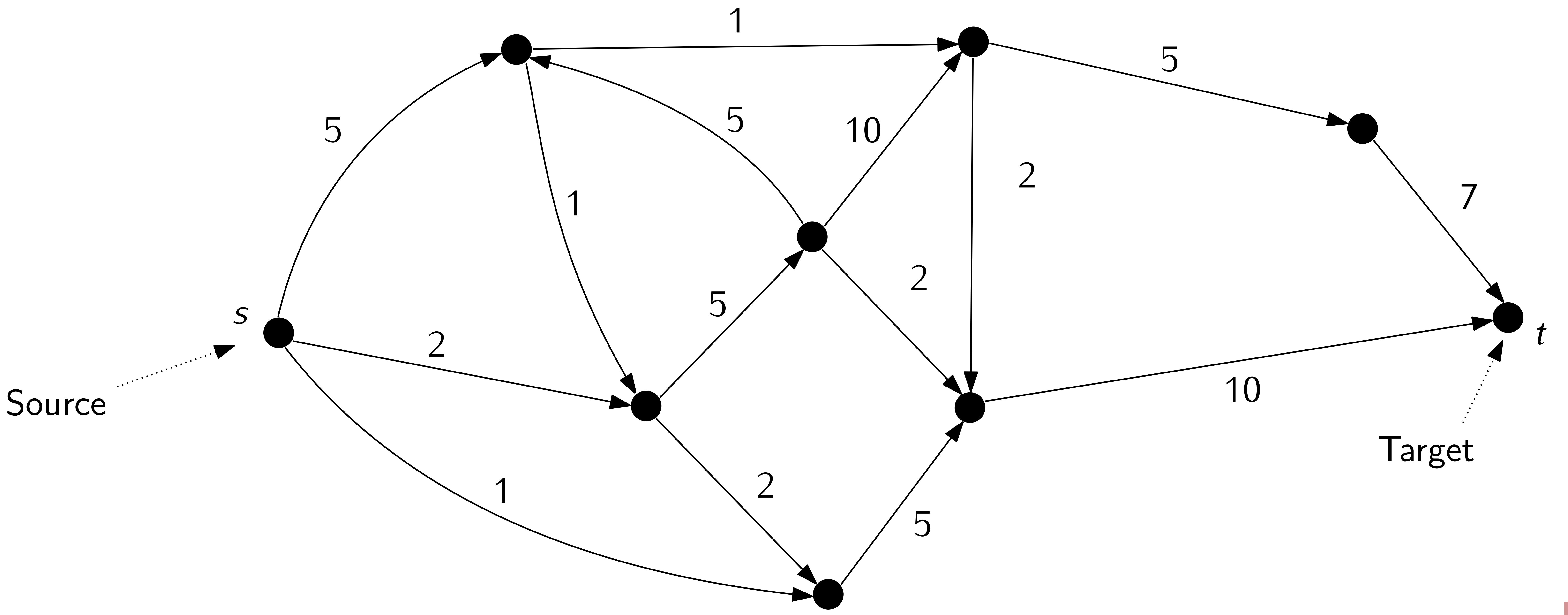
A flow network is a directed graph without isolated vertices and:

- a source s
- a target t



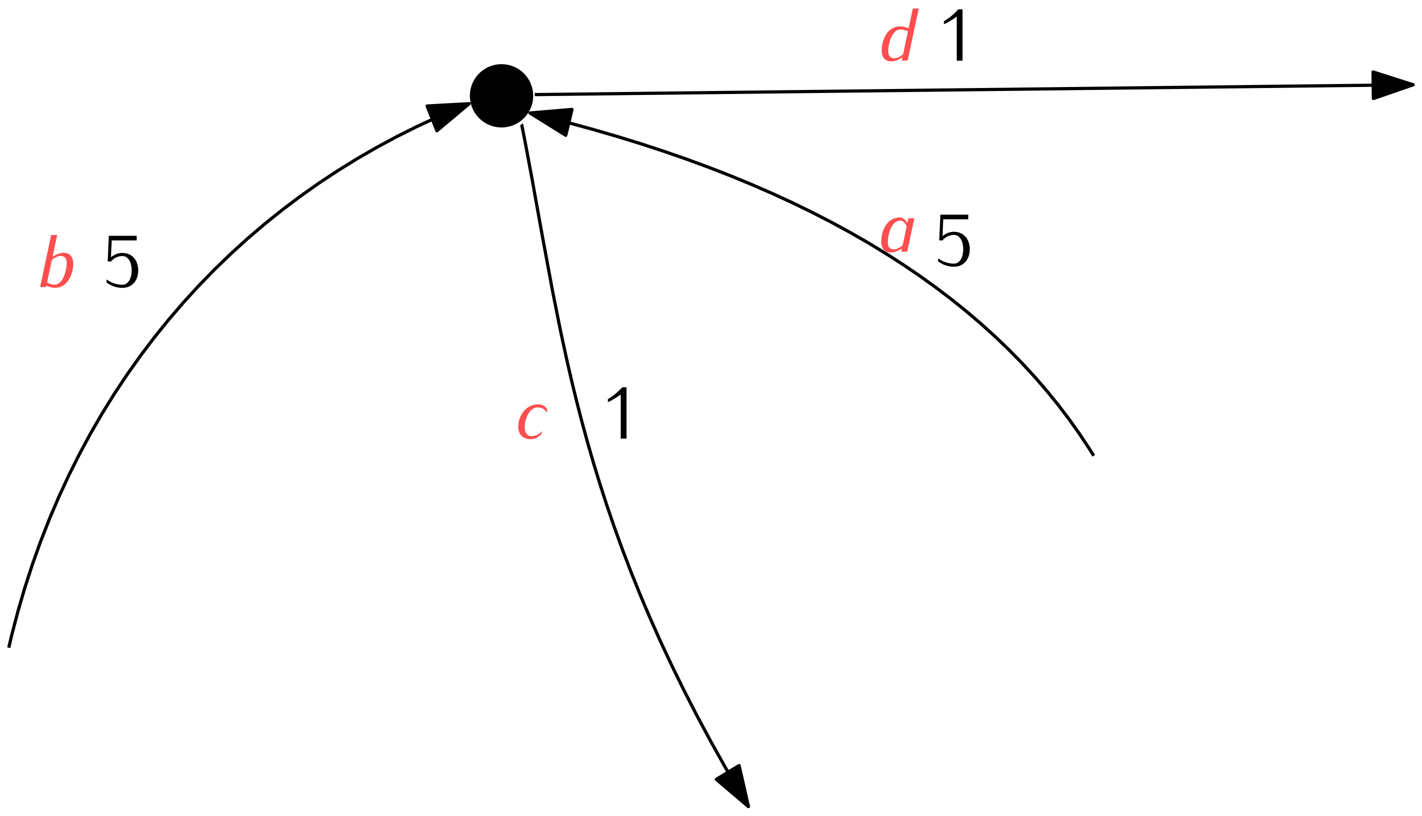
A flow network is a directed graph without isolated vertices and:

- a source s
- a target t
- capacities: a function $c\colon E \to \mathbb{N}$



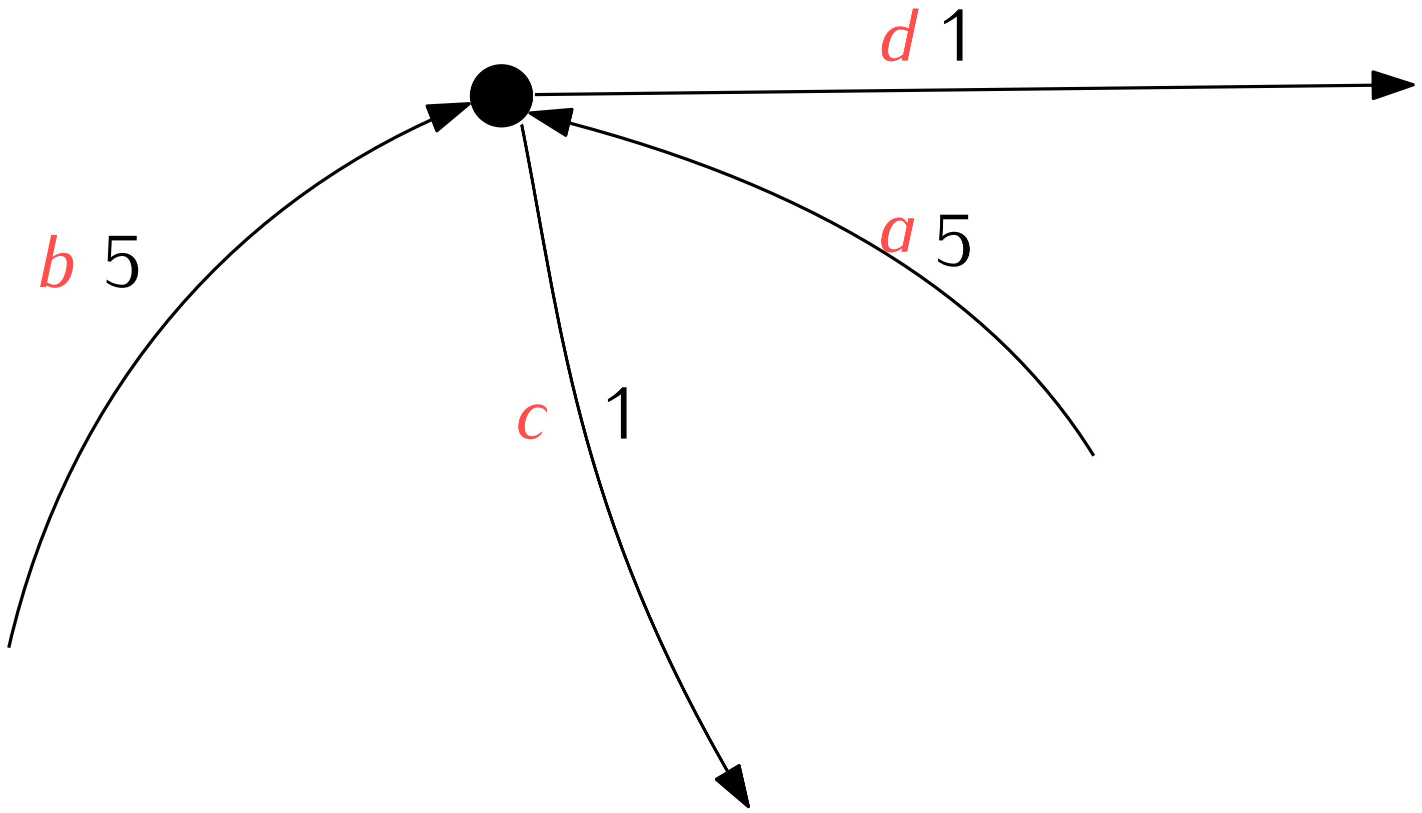
A flow network is a directed graph without isolated vertices and:

- a source s
- a target t



A flow is a function $f\colon E \to \mathbb{R}_{\geq 0}$ satisfying:

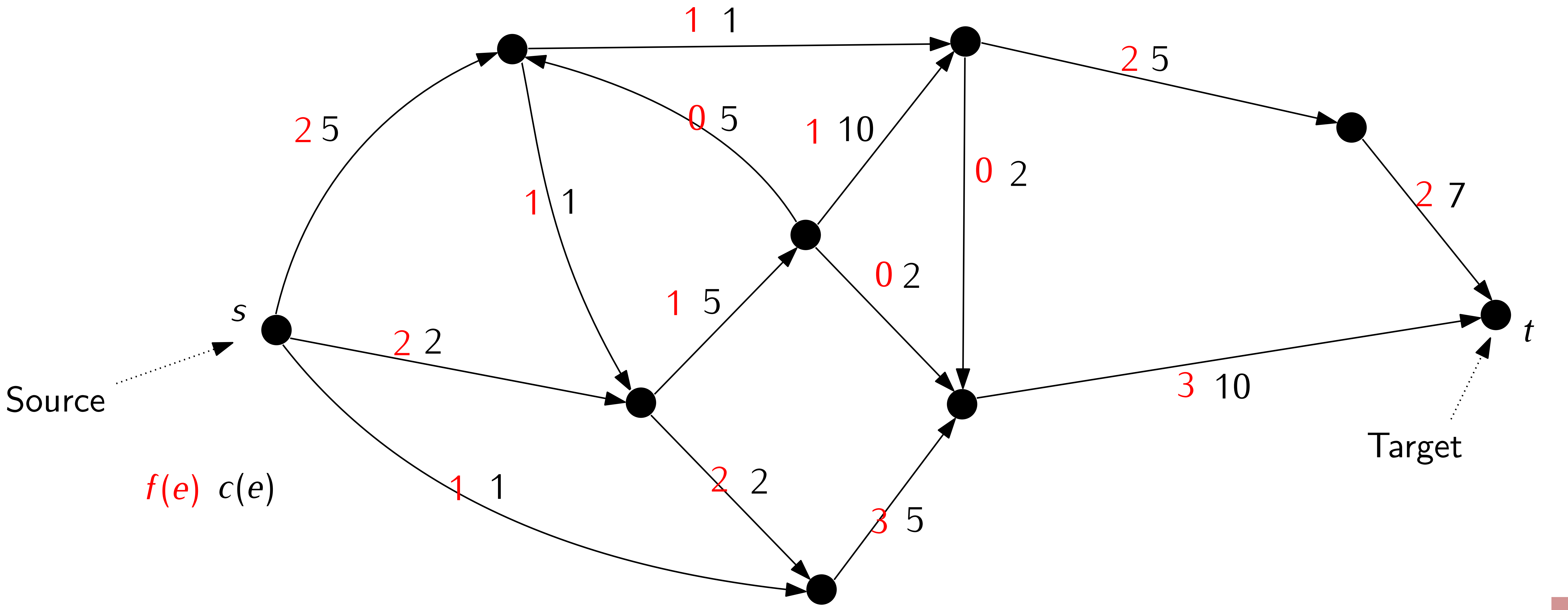
- For all $e \in E$, $0 \leq f(e) \leq c(e)$
- For all $v \in V \setminus \{s, t\}$, $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.



$f(e)$ $c(e)$

A flow is a function $f\colon E \to \mathbb{R}_{\geq 0}$ satisfying:

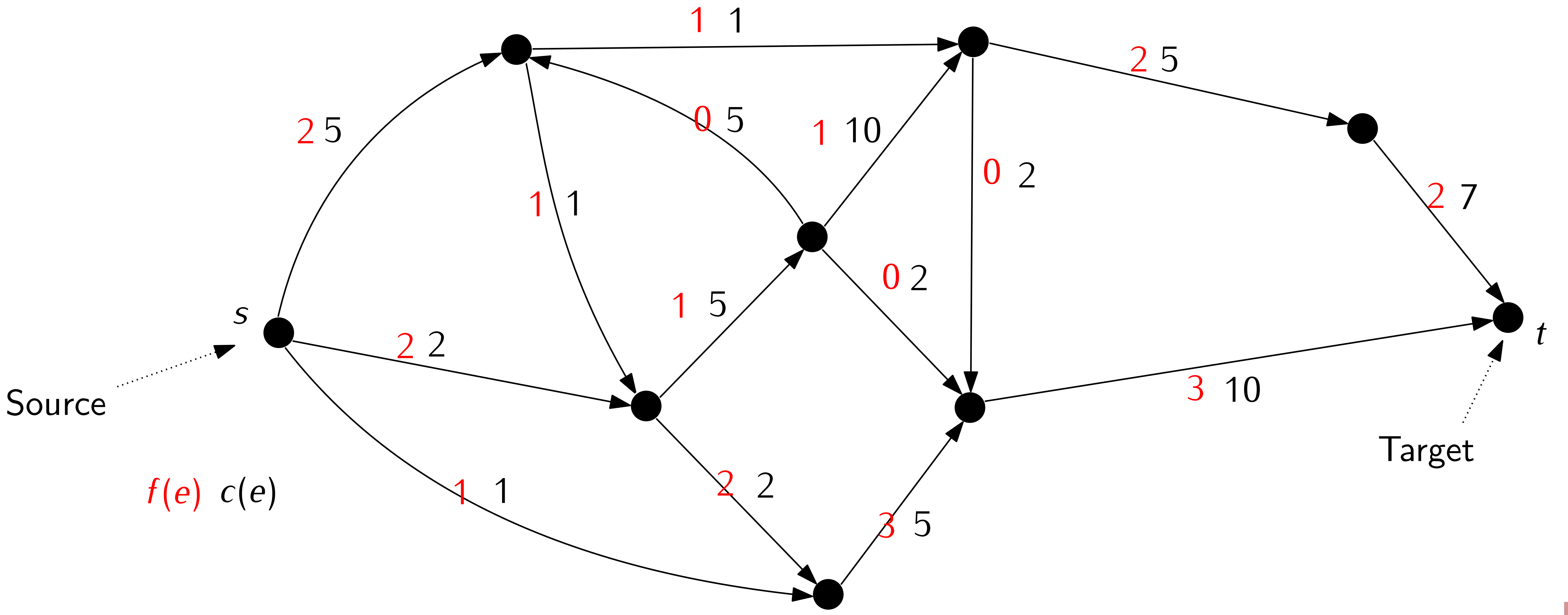
- For all $e \in E$, $0 \leq f(e) \leq c(e)$
- For all $v \in V \setminus \{s, t\}$, $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.



A flow is a function $f\colon E \to \mathbb{R}_{\geq 0}$ satisfying:

- For all $e \in E$, $0 \leq f(e) \leq c(e)$
- For all $v \in V \setminus \{s, t\}$, $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.

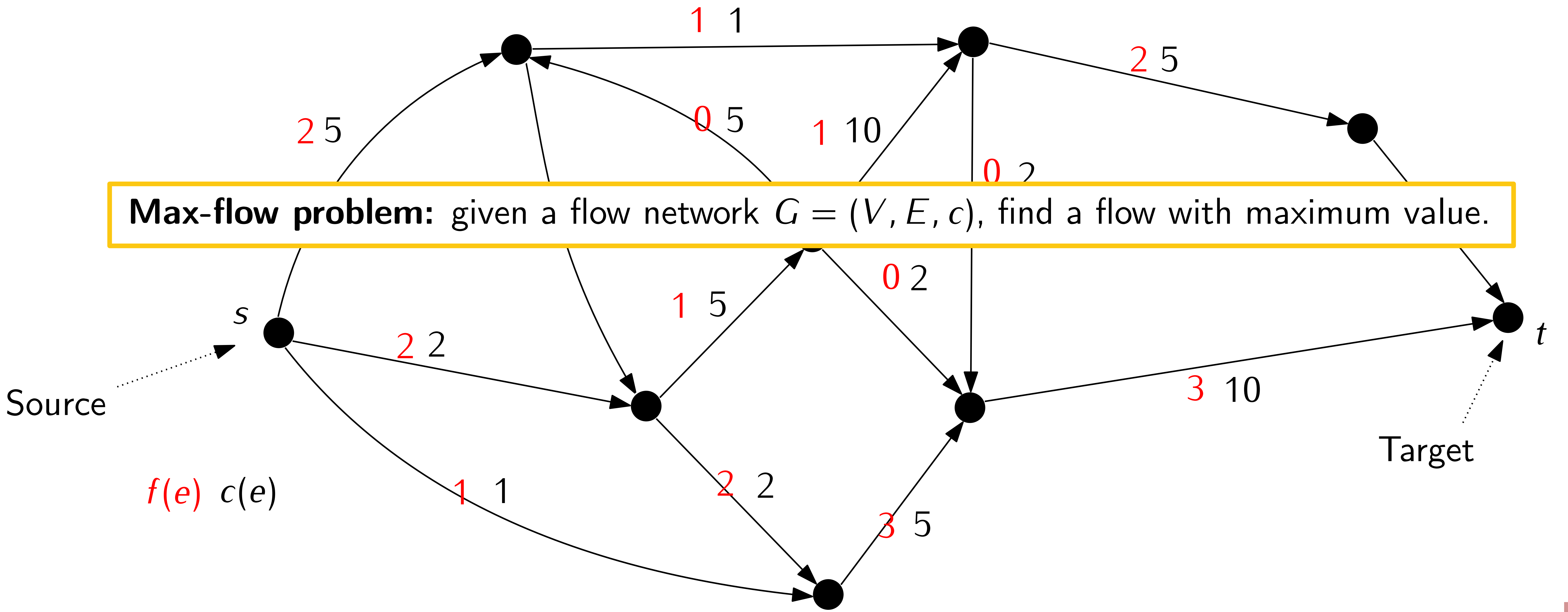
Value of f: $\sum_{e \text{ out of } s} f(e)$



A flow is a function $f\colon E \to \mathbb{R}_{\geq 0}$ satisfying:

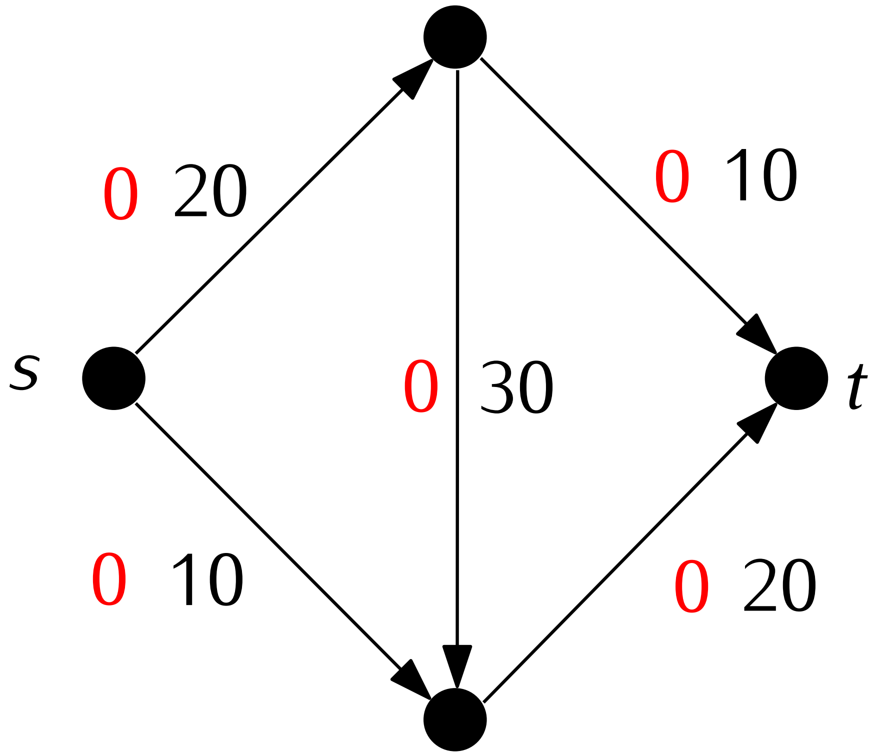
- For all $e \in E$, $0 \leq f(e) \leq c(e)$
- For all $v \in V \setminus \{s, t\}$, $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.

Value of f: $\sum_{e \text{ out of } s} f(e)$



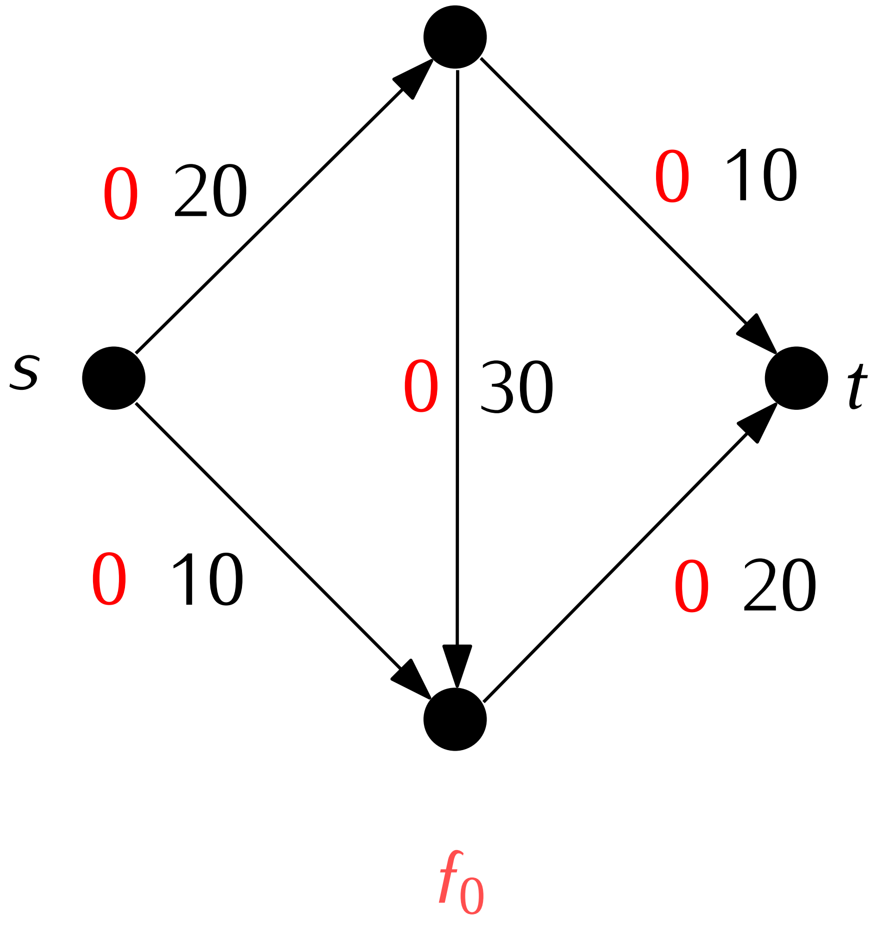
Max-flow problem: given a flow network $G = (V, E, c)$, find a flow with maximum value.

- Start with 0 flow: $f(e) = 0$ for all $e \in E$

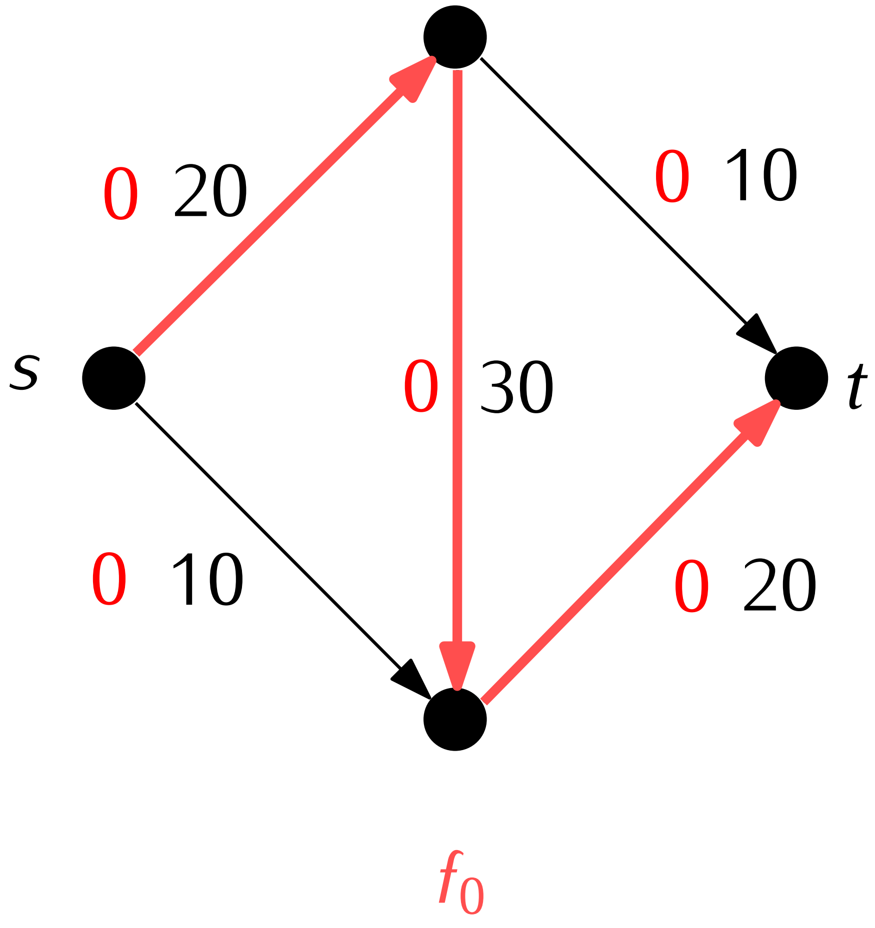


f_0

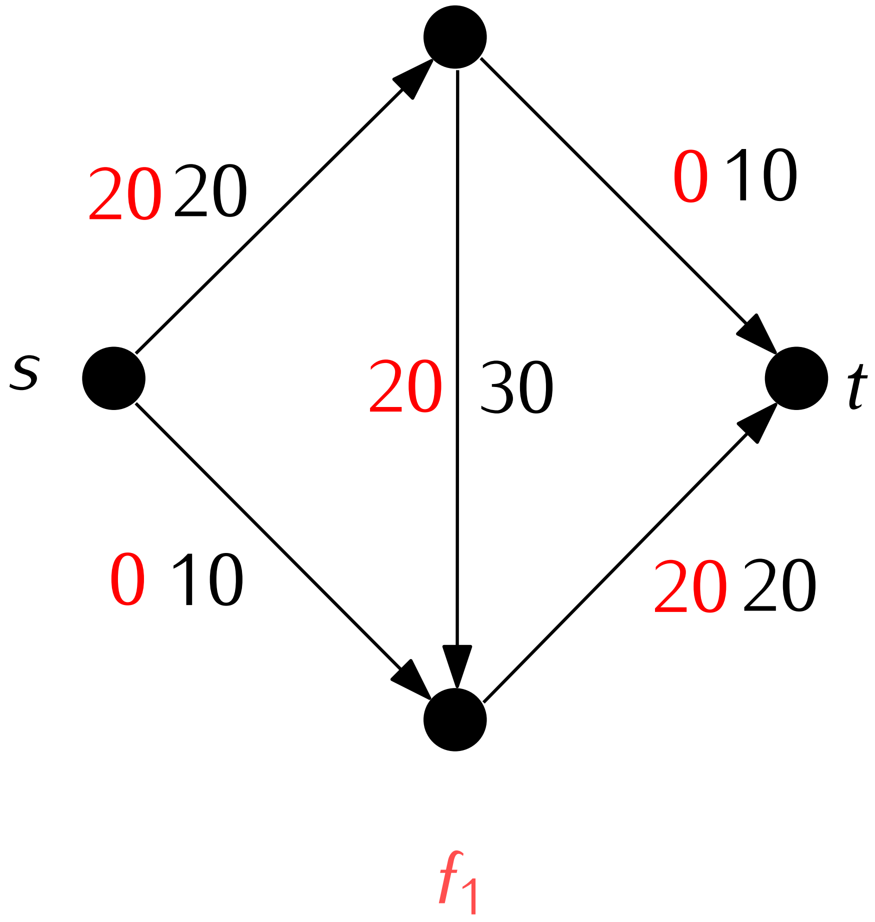
- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...



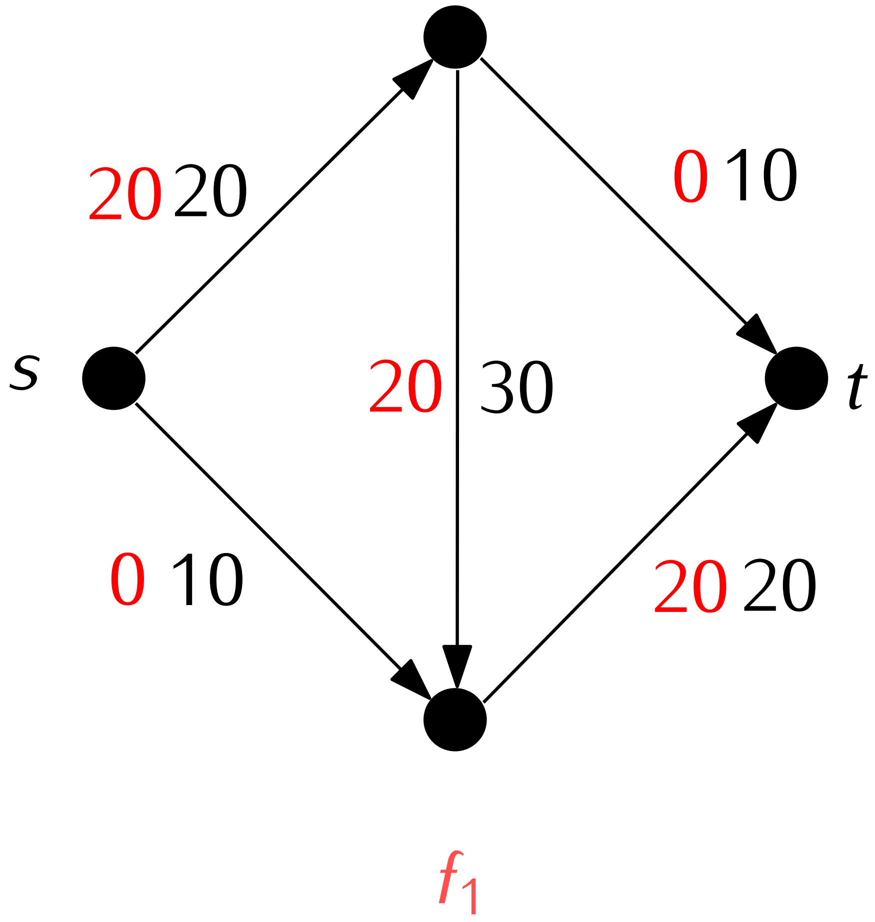
- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...
- "Downstream step": can increase the value along any s-t path that is not saturated



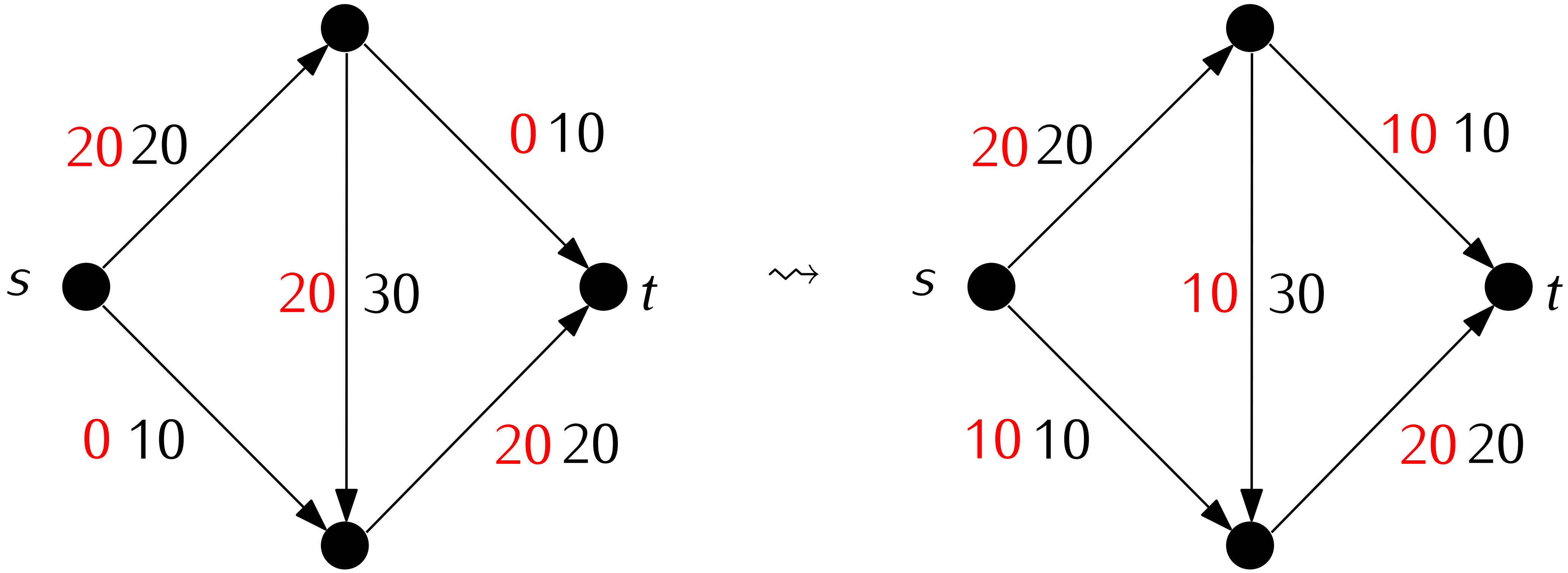
- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...
- "Downstream step": can increase the value along any s-t path that is not saturated



- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...
- "Downstream step": can increase the value along any s-t path that is not saturated
- "Upstream step": one may go backwards along an edge and redirect some flow

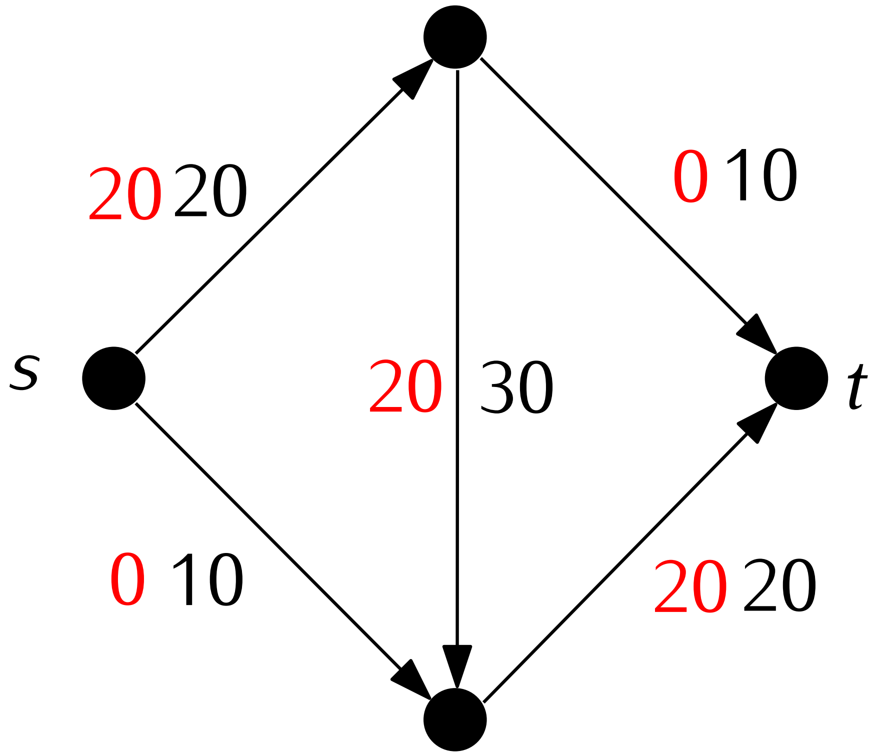


- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...
- "Downstream step": can increase the value along any s-t path that is not saturated
- "Upstream step": one may go backwards along an edge and redirect some flow



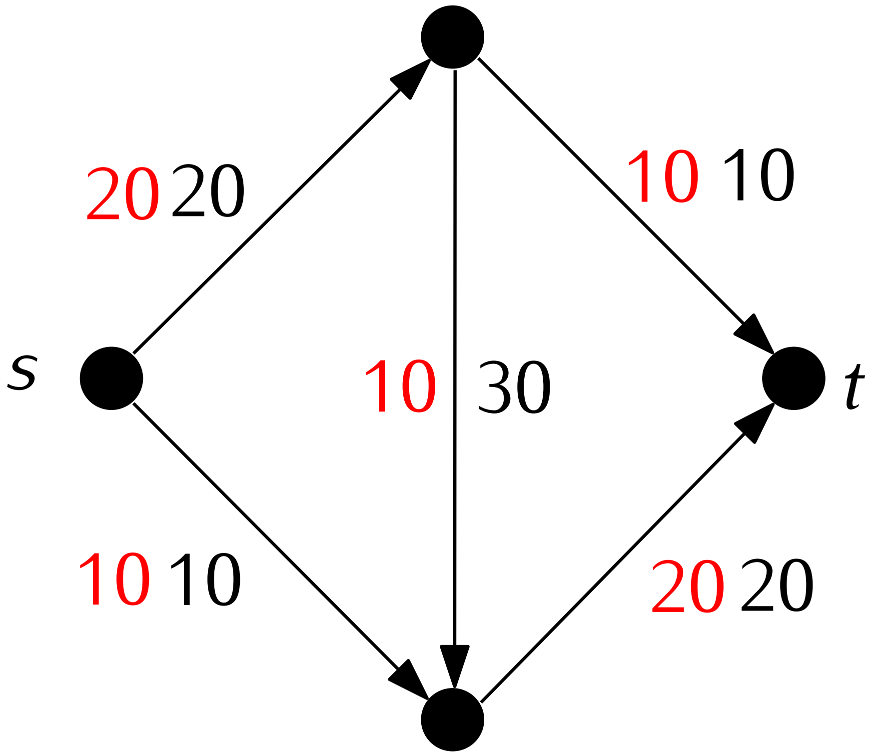
f_1 f_2

- Start with 0 flow: $f(e) = 0$ for all $e \in E$
- Find a way to improve the flow...
- "Downstream step": can increase the value along any s-t path that is not saturated
- "Upstream step": one may go backwards along an edge and redirect some flow
- **Want:** data structure to find those steps efficiently



f_1

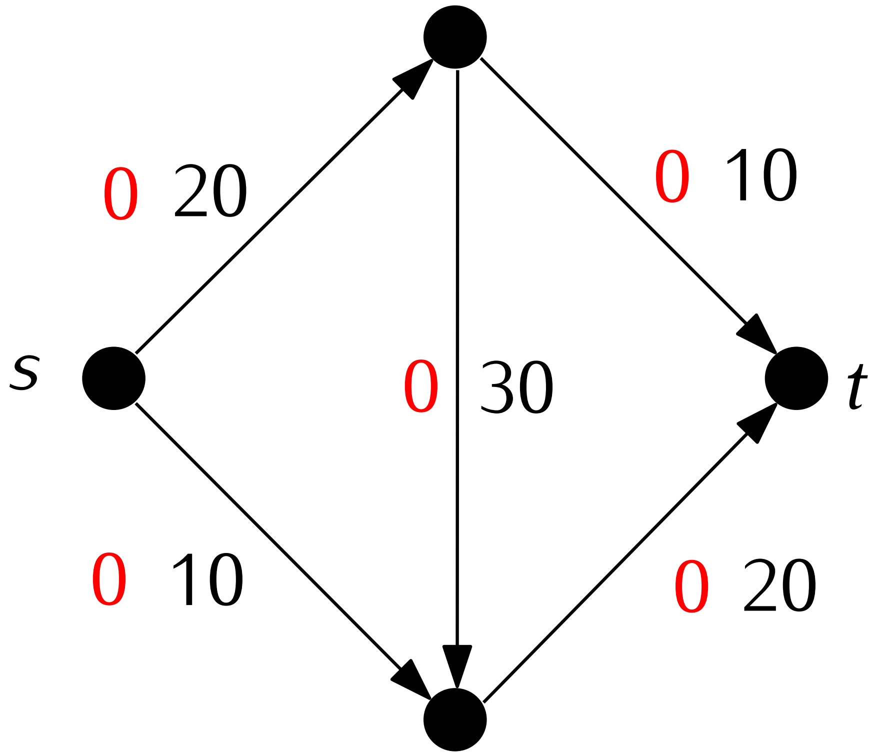
$\rightsquigarrow$



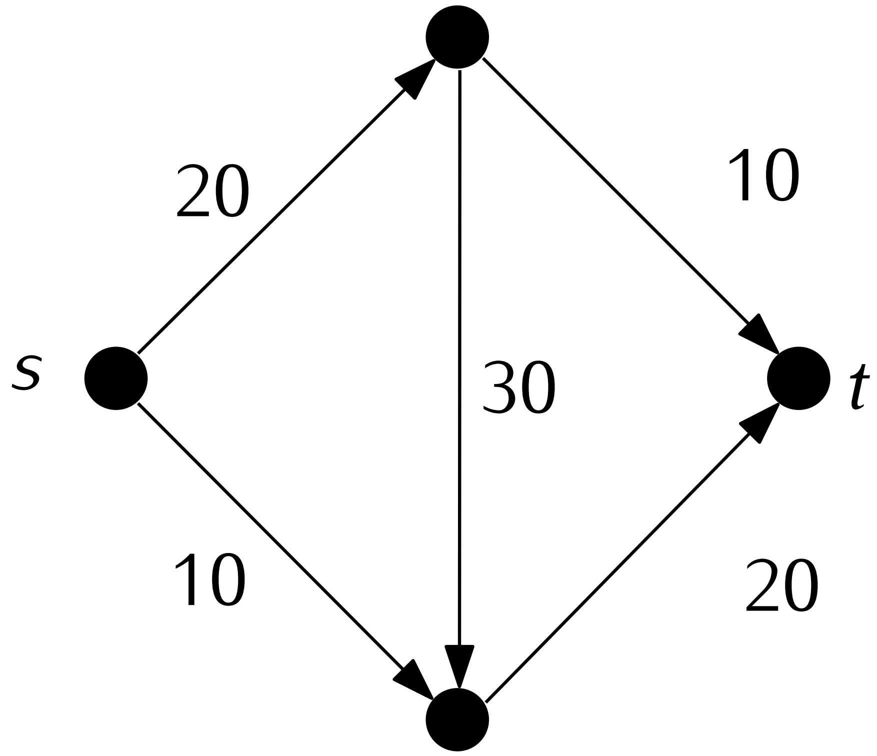
f_2

Definition (Residual network). $G = (V, E, c)$ a flow network, $f \colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
-



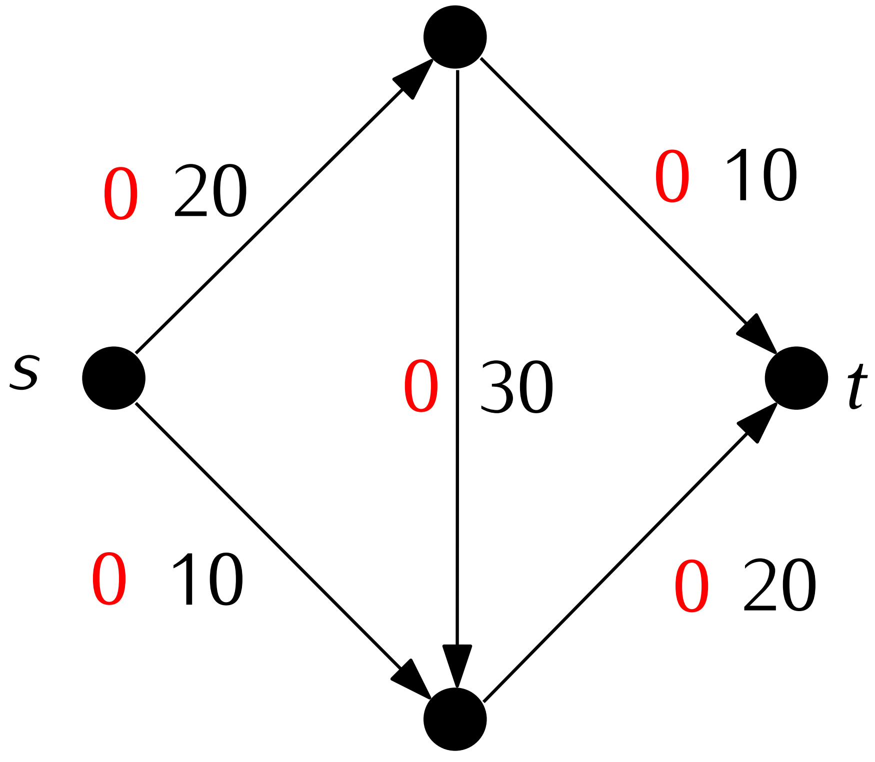
f_0



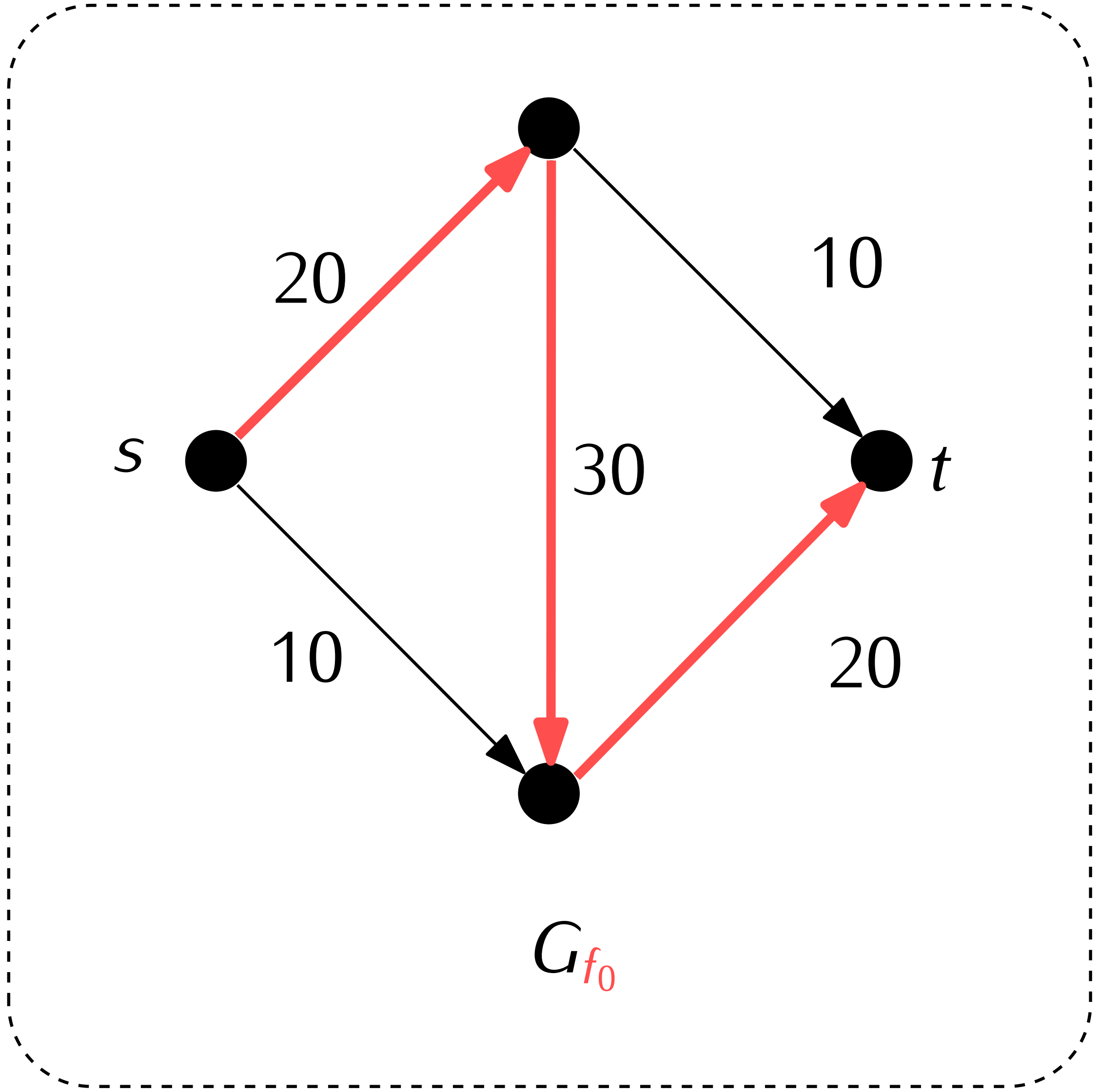
G_{f_0}

Definition (Residual network). $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
-



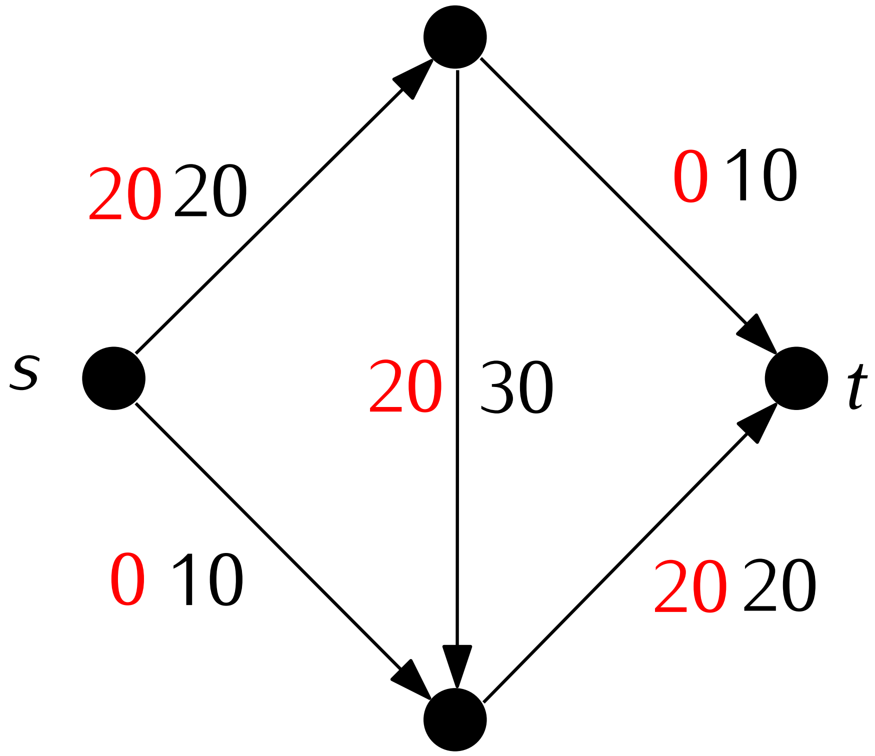
f_0



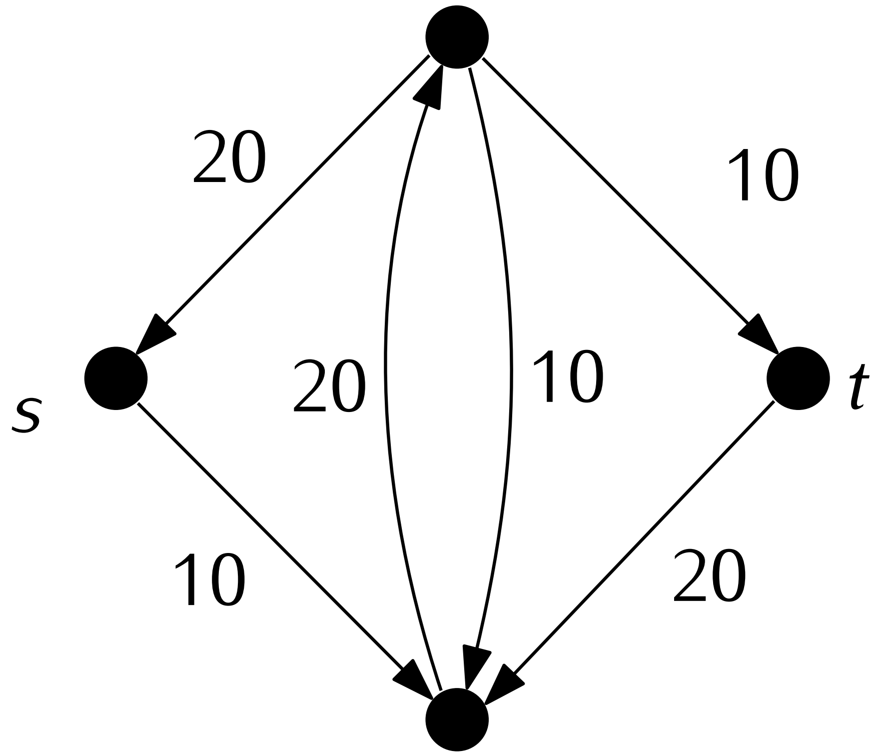
G_{f_0}

Definition (Residual network). $G = (V, E, c)$ a flow network, $f: E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)



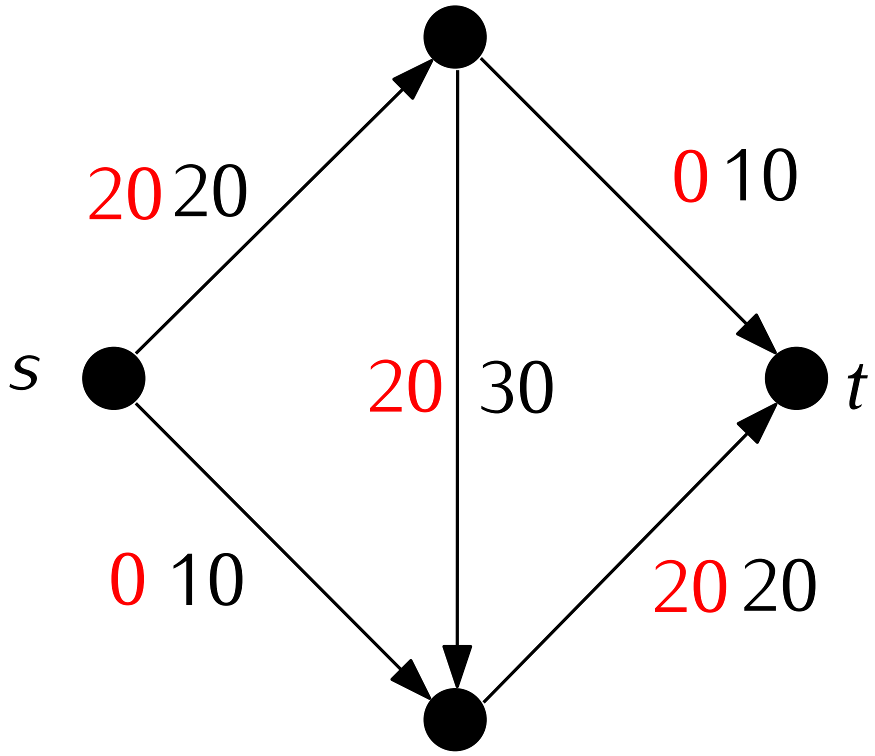
f_1



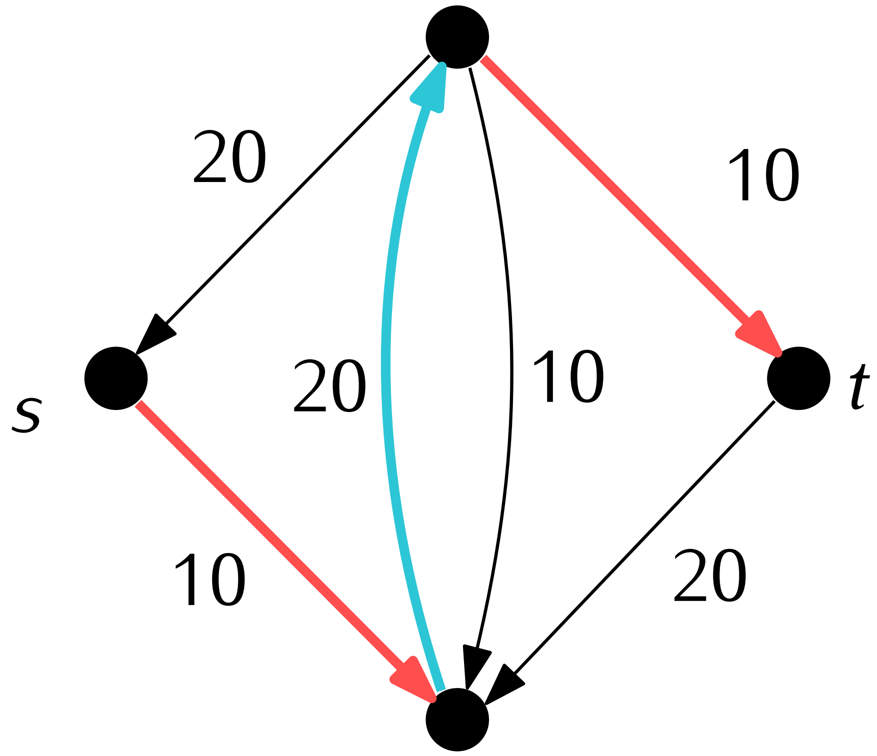
G_{f_1}

Definition (Residual network). $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)



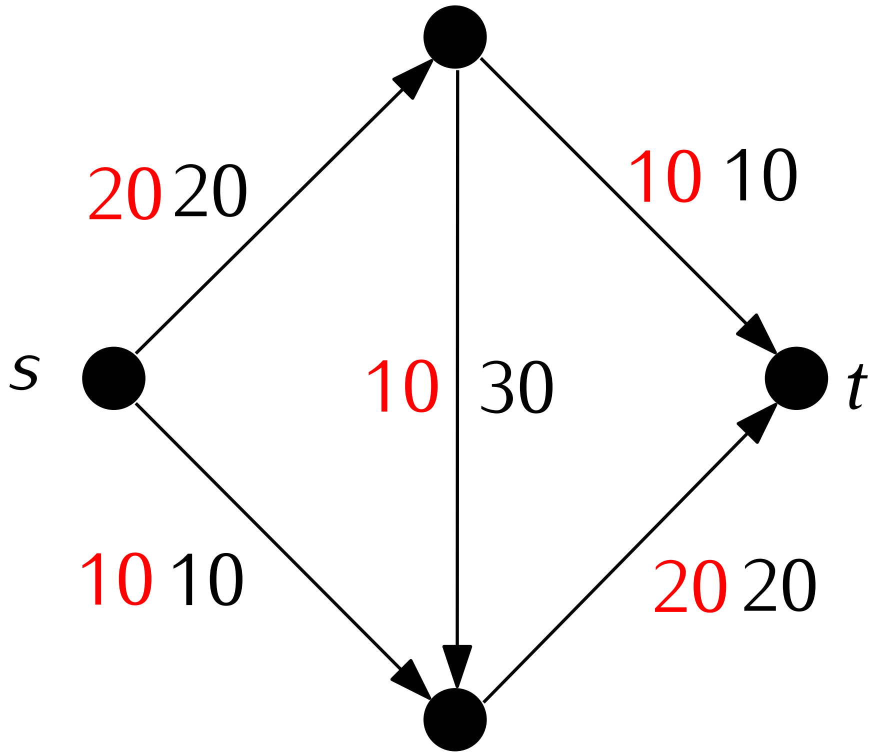
f_1



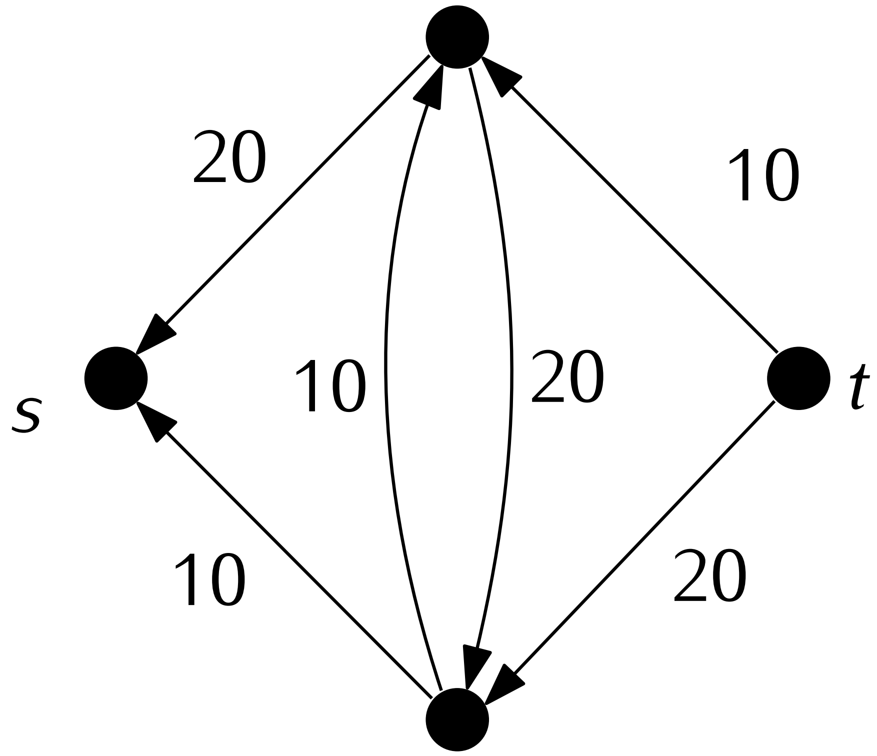
G_{f_1}

Definition (Residual network). $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)



f_2



G_{f_2}

Definition (Residual network). $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:
   (a) Find $m = \min\{c_f(e_i) \mid i \in \{1, \ldots, k\}\}$
   (b) Define:
      - $f'(e_i) := f(e_i) + m$ for all $i \in \{0, \ldots, k\}$ if $e_i \in E$
      - $f'(e_i) := f(e_i) - m$ if $\overleftarrow{e_i}$ is in E
      - $f'(e) = f(e)$ for other edges
   (c) $f := f'$
3. Return f

Definition (Residual network)**.** $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:
   (a) Find $m = \min\{c_f(e_i) \mid i \in \{1, \ldots, k\}\}$
   (b) Define:
      - $f'(e_i) := f(e_i) + m$ for all $i \in \{0, \ldots, k\}$ if $e_i \in E$
      - $f'(e_i) := f(e_i) - m$ if $\overleftarrow{e_i}$ is in E
      - $f'(e) = f(e)$ for other edges
   (c) $f := f'$

   $\Bigg\}$ $\textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Definition (Residual network). $G = (V, E, c)$ a flow network, $f\colon E \to \mathbb{R}_{\geq 0}$ a flow
Define $G_f = (V, E_f, c_f)$ by:

- For $e \in E$ where $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$ (downstream step)
- For $e = (u, v) \in E$ where $f(e) > 0$, then $\overleftarrow{e} = (v, u) \in E_f$ and $c_f(\overleftarrow{e}) = f(e)$ (upstream step)

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:
   (a) Find $m = \min\{c_f(e_i) \mid i \in \{1, \ldots, k\}\}$
   (b) Define:
      - $f'(e_i) := f(e_i) + m$ for all $i \in \{0, \ldots, k\}$ if $e_i \in E$
      - $f'(e_i) := f(e_i) - m$ if $\overleftarrow{e_i}$ is in E
      - $f'(e) = f(e)$ for other edges
   (c) $f := f'$

   $\Big\}\ \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Questions:

- Does it terminate? 2. could loop forever...
- How fast does it terminate?
- Is f a flow?
- Does f always have maximal value?

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

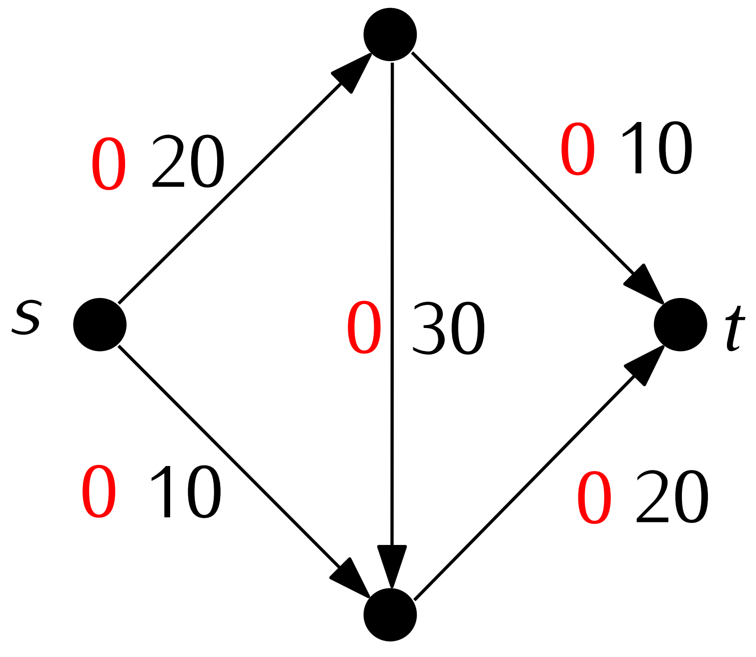
Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Algorithm: on input $G = (V, E, c)$,

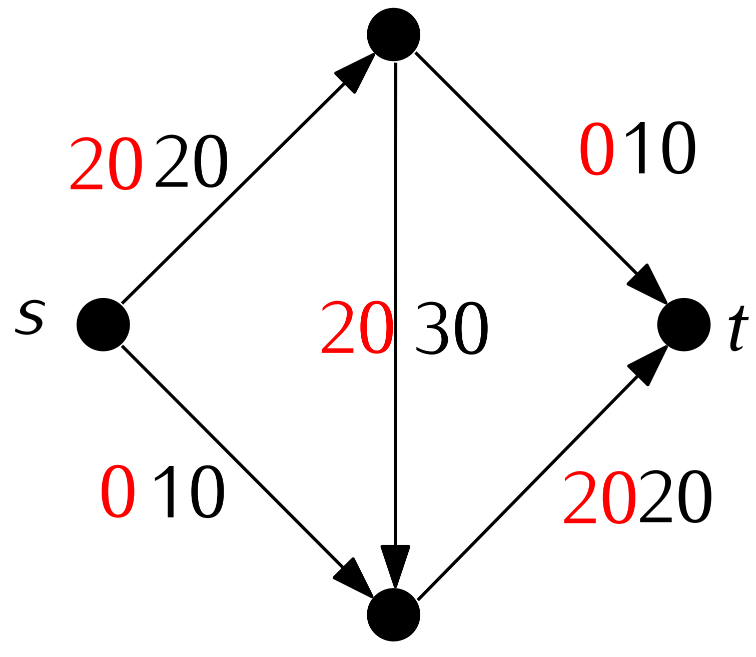
1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

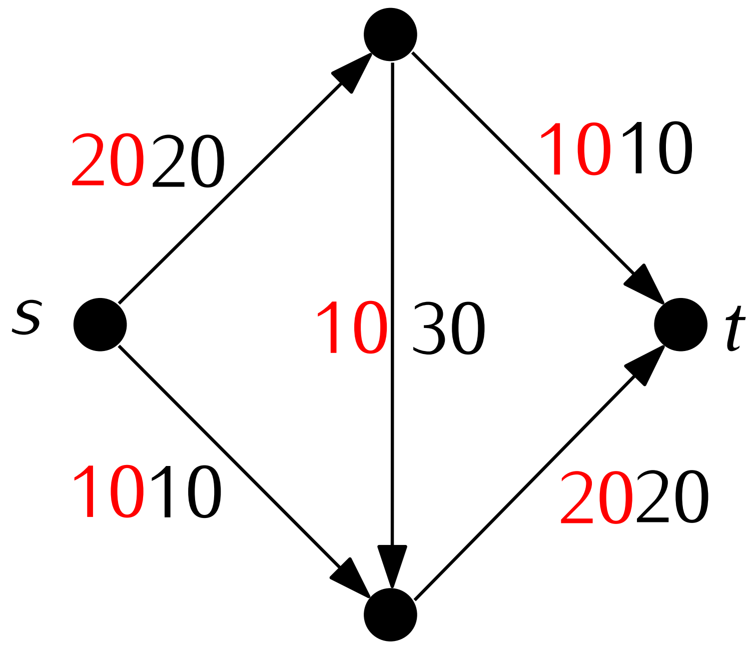
Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration



f_0



f_1



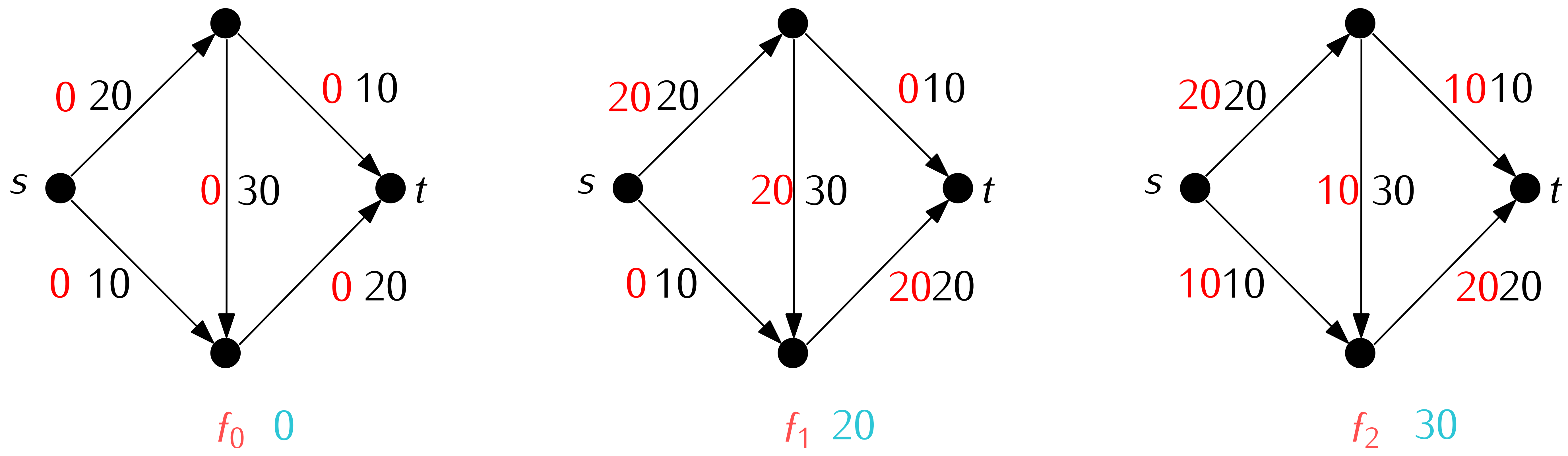
f_2

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \dots, e_k$ in G_f:
   $f := \textsc{Augment}(f, (e_1, \dots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Attempt: $m(f) = \sum_{e \text{ out of } s} f(e)$



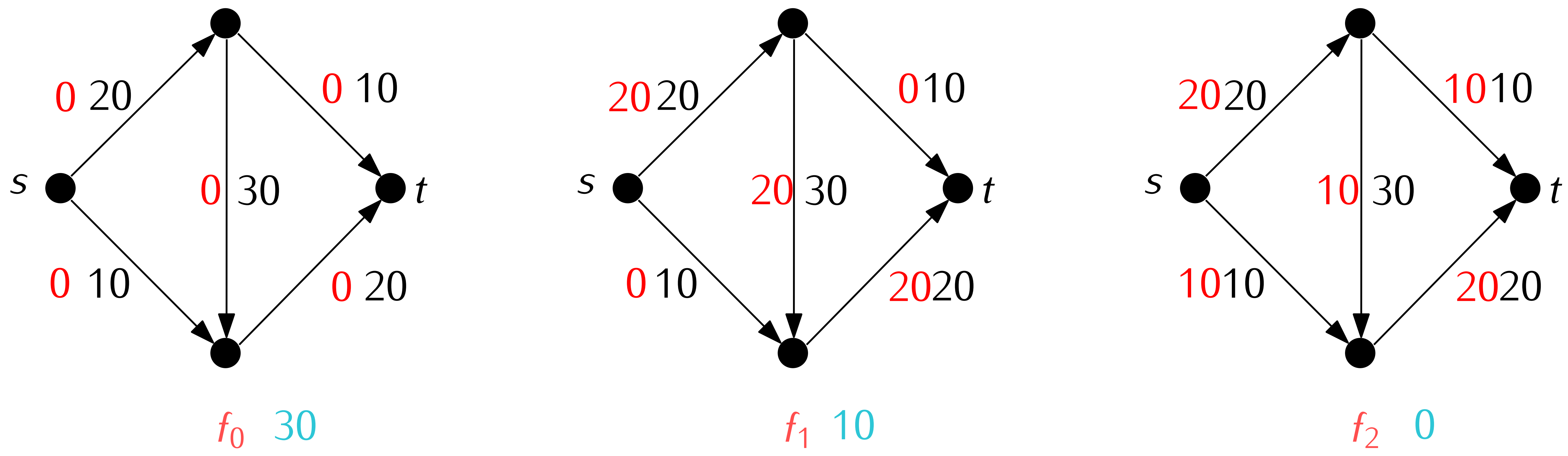
Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Attempt: $m(f) = \sum_{e \text{ out of } s} (c(e) - f(e))$



Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Attempt: $m(f) = \sum_{e \text{ out of } s}(c(e) - f(e))$ ✅

Lemma. $m(f) \geq 0$ and strictly decreases at every iteration.

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \dots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \dots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Attempt: $m(f) = \sum_{e \text{ out of } s}(c(e) - f(e))$ ✅

Lemma. $m(f) \geq 0$ and strictly decreases at every iteration.

- Algorithm terminates in at most $C := \sum_{e \text{ out of } s} c(e)$ iterations.

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:
   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

Attempt: $m(f) = \sum_{e \text{ out of } s}(c(e) - f(e))$ ✅

Lemma. $m(f) \geq 0$ and strictly decreases at every iteration.

- Algorithm terminates in at most $C := \sum_{e \text{ out of } s} c(e)$ iterations.
- $O(C|E|)$ running time

Algorithm: on input $G = (V, E, c)$,

1. Start with zero flow: $f(e) = 0$ for all $e \in E$
2. While there exists a simple s-t path $e_1, \ldots, e_k$ in G_f:

   $f := \textsc{Augment}(f, (e_1, \ldots, e_k))$
3. Return f

Usual idea: find a measure $m(f) \in \mathbb{N}$ that decreases at every iteration

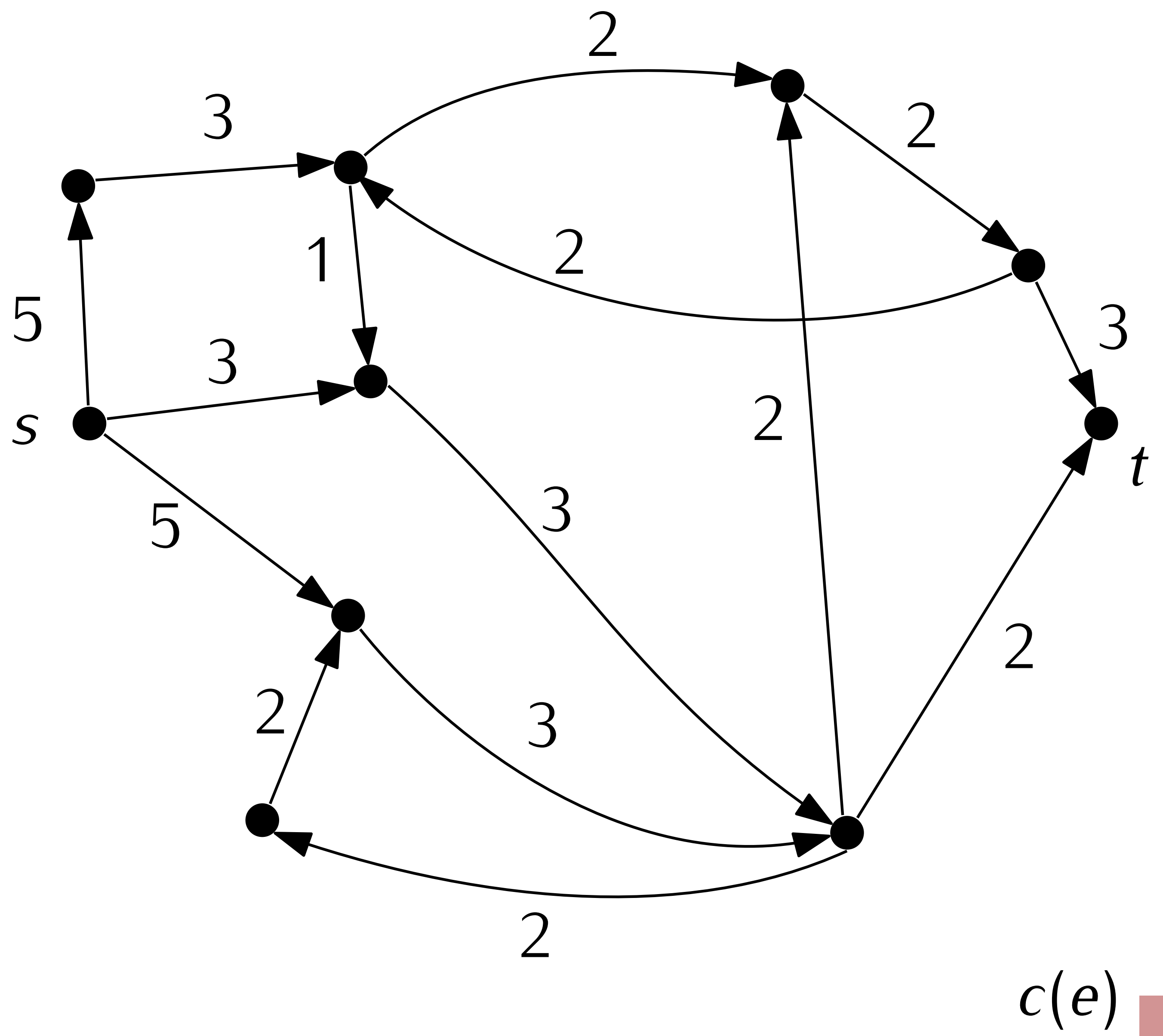
Attempt: $m(f) = \sum_{e \text{ out of } s}(c(e) - f(e))$ ✅

Lemma. $m(f) \geq 0$ and strictly decreases at every iteration.

- Algorithm terminates in at most $C := \sum_{e \text{ out of } s} c(e)$ iterations.
- $O(C|E|)$ running time
- We later see that there are at most $|V||E|$ iterations if we choose shortest path at 2.

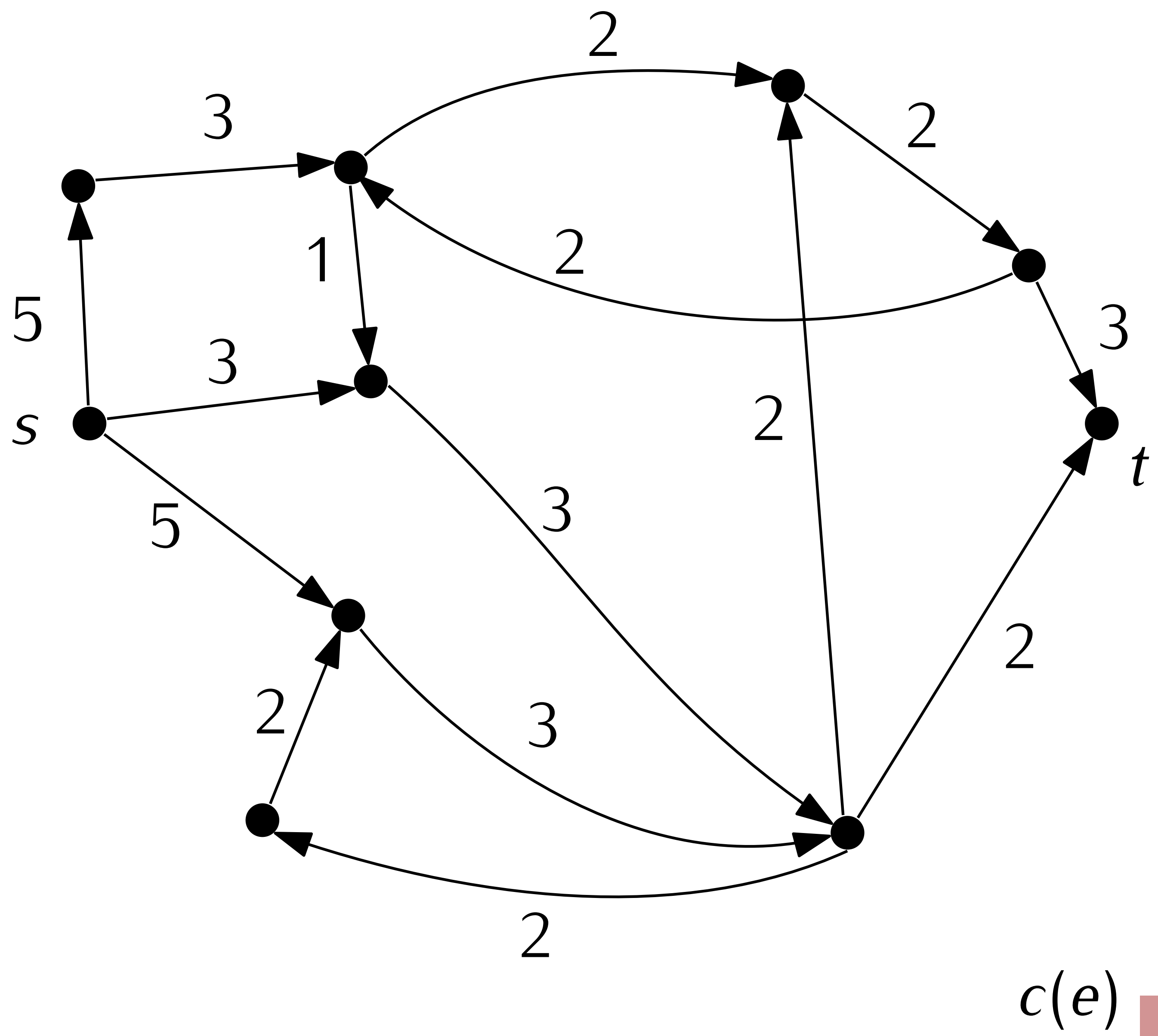
  $\rightsquigarrow$ Edmonds-Karp algorithm

What are values that can in principle be achieved?



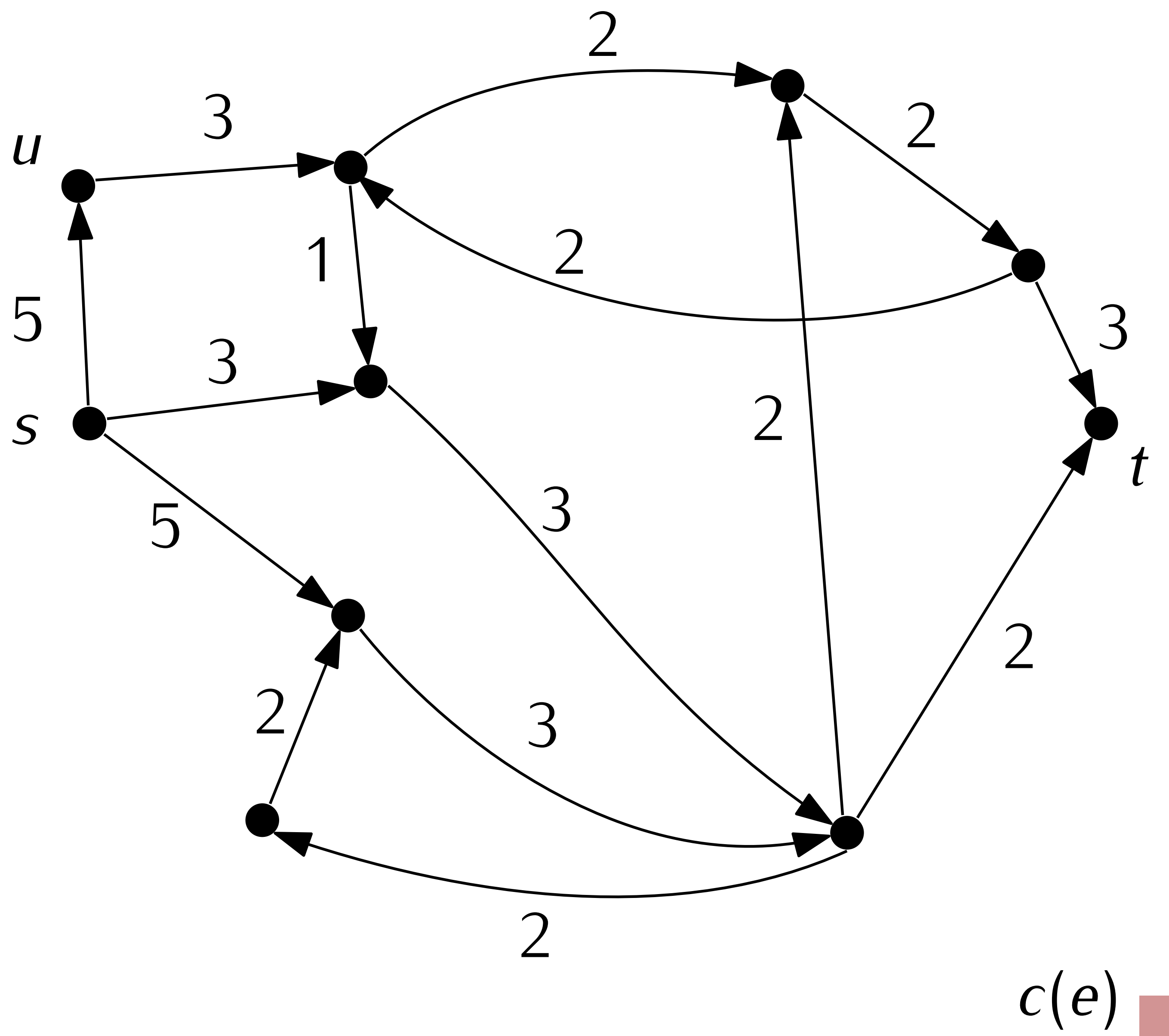
What are values that can in principle be achieved?

- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$



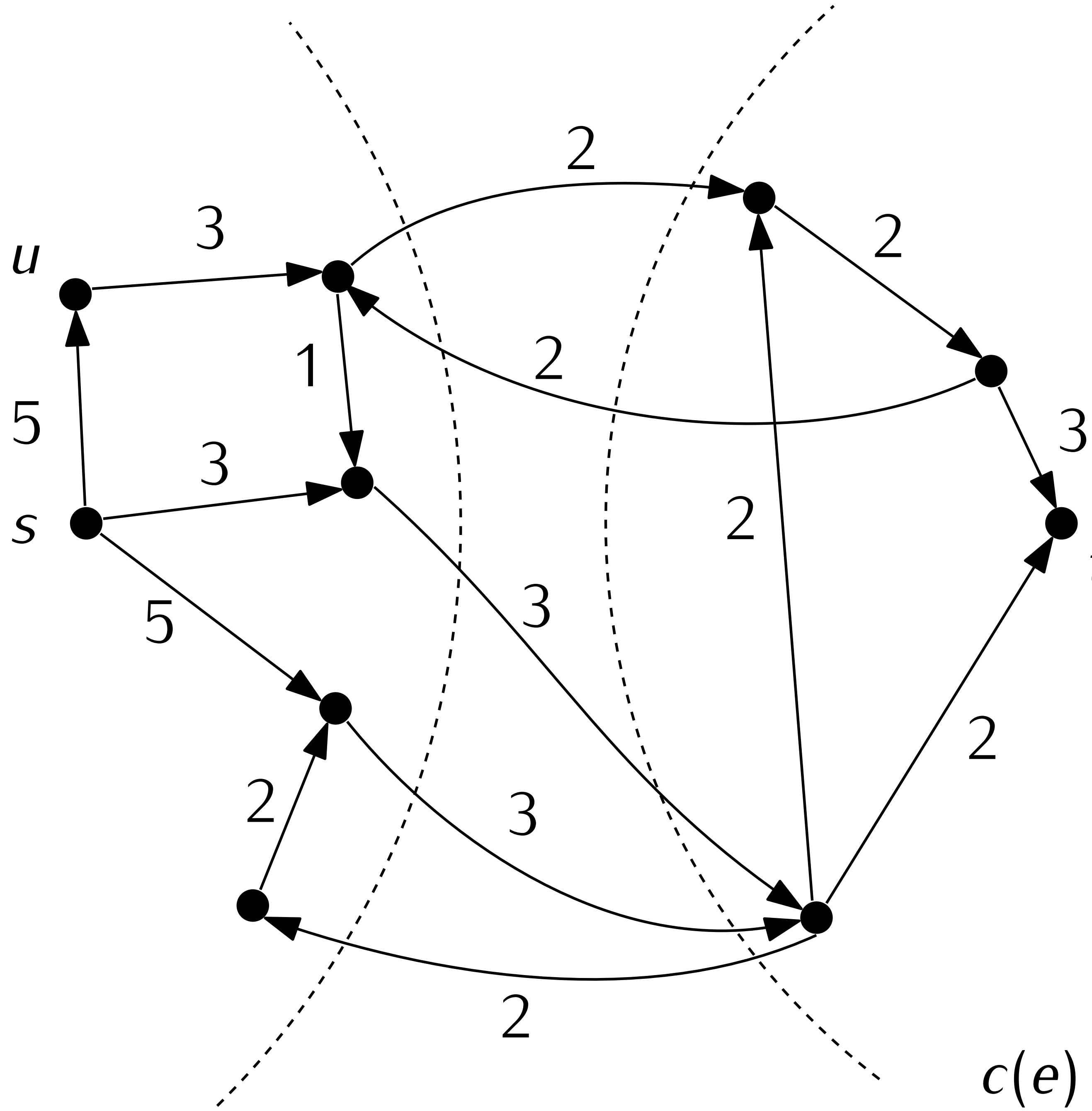
What are values that can in principle be achieved?

- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$
- Value is $\leq \sum_{e \text{ out of } \{s,u\}} c(e) = 11$



What are values that can in principle be achieved?

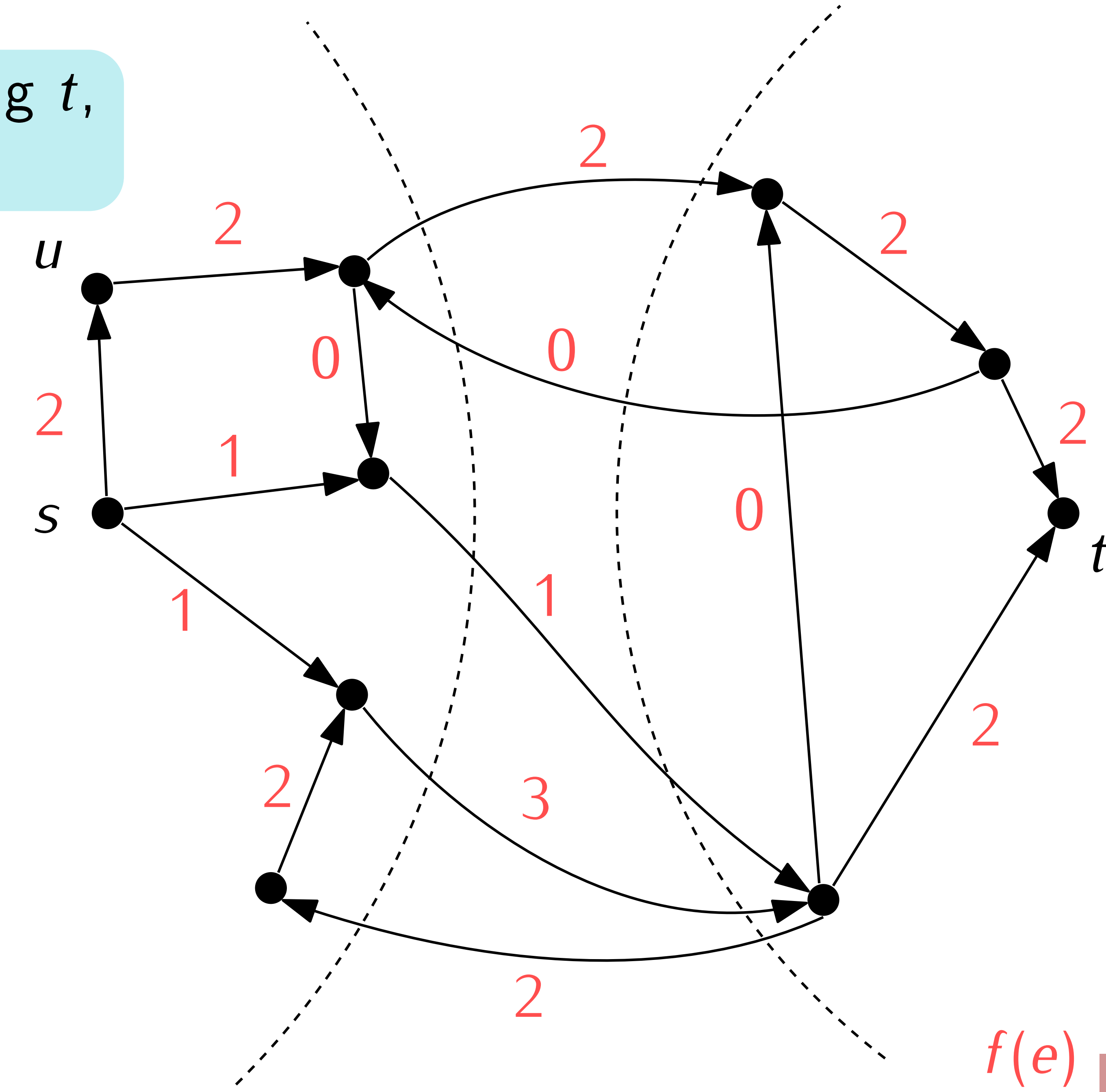
- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$
- Value is $\leq \sum_{e \text{ out of } \{s,u\}} c(e) = 11$
- Value is ≤ 8



What are values that can in principle be achieved?

- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$
- Value is $\leq \sum_{e \text{ out of } \{s,u\}} c(e) = 11$
- Value is ≤ 8

Lemma. For any $A \subseteq V$ containing s, $B := V \setminus A$ containing t, the value of f is equal to $\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$.

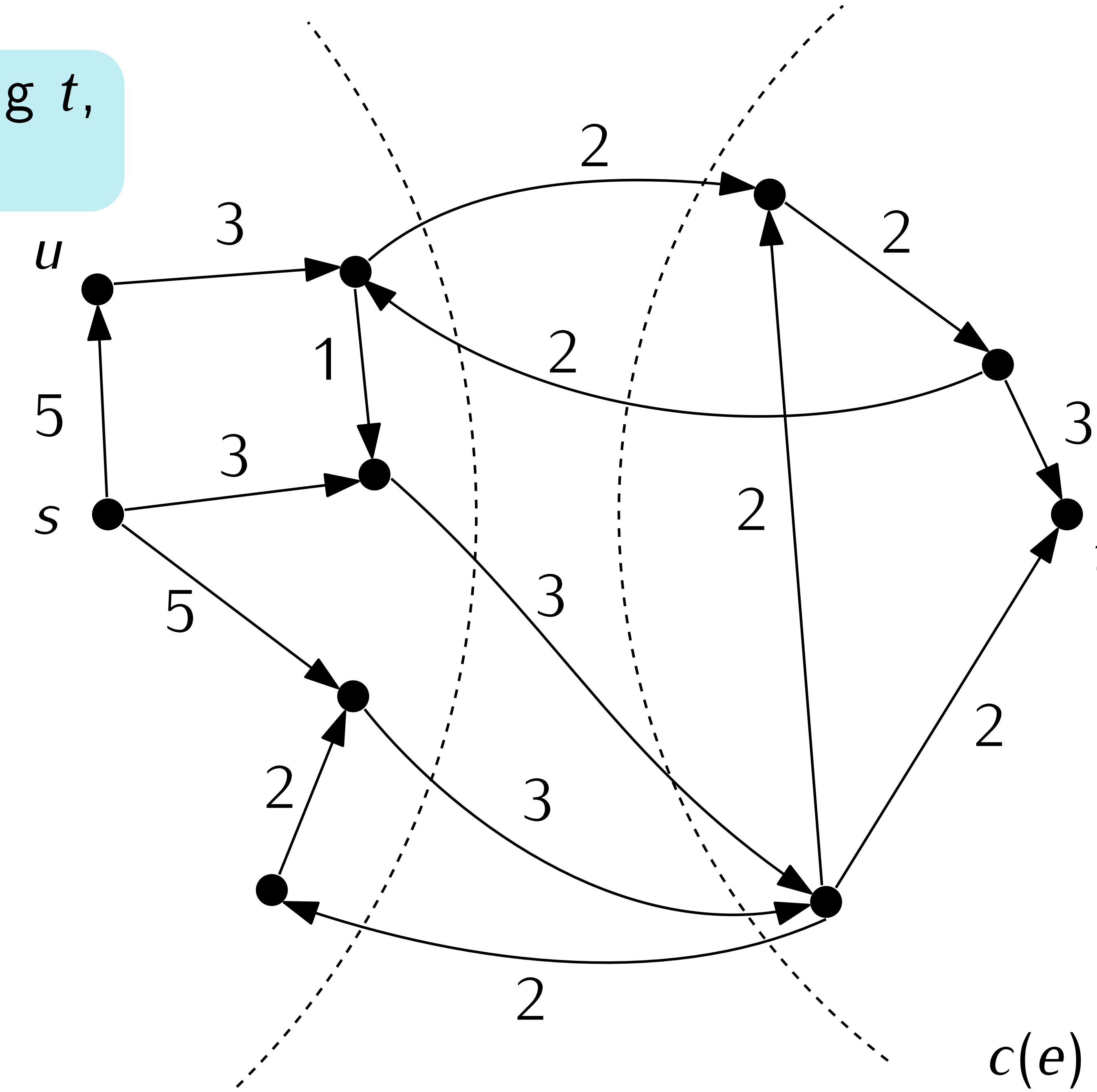


What are values that can in principle be achieved?

- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$
- Value is $\leq \sum_{e \text{ out of } \{s,u\}} c(e) = 11$
- Value is ≤ 8

Lemma. For any $A \subseteq V$ containing s, $B := V \setminus A$ containing t, the value of f is equal to $\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$.

$\rightsquigarrow$ the maximal value of a flow is at most $\sum_{e \text{ out of } A} c(e)$.



What are values that can in principle be achieved?

- Value is $\leq \sum_{e \text{ out of } s} c(e) = 13$
- Value is $\leq \sum_{e \text{ out of } \{s,u\}} c(e) = 11$
- Value is ≤ 8

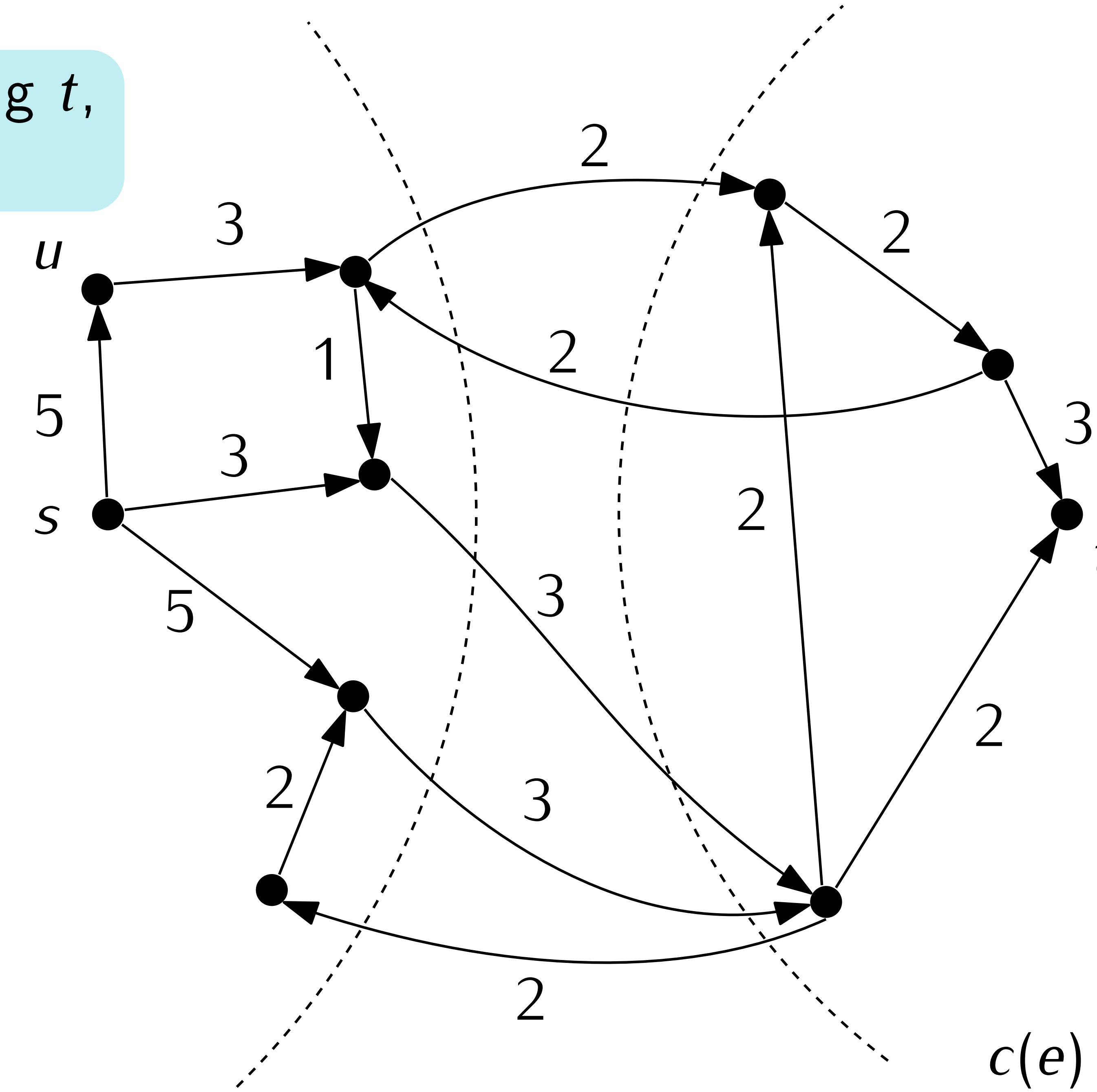
Lemma. For any $A \subseteq V$ containing s, $B := V \setminus A$ containing t, the value of f is equal to $\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$.

$\rightsquigarrow$ the maximal value of a flow is at most $\sum_{e \text{ out of } A} c(e)$.

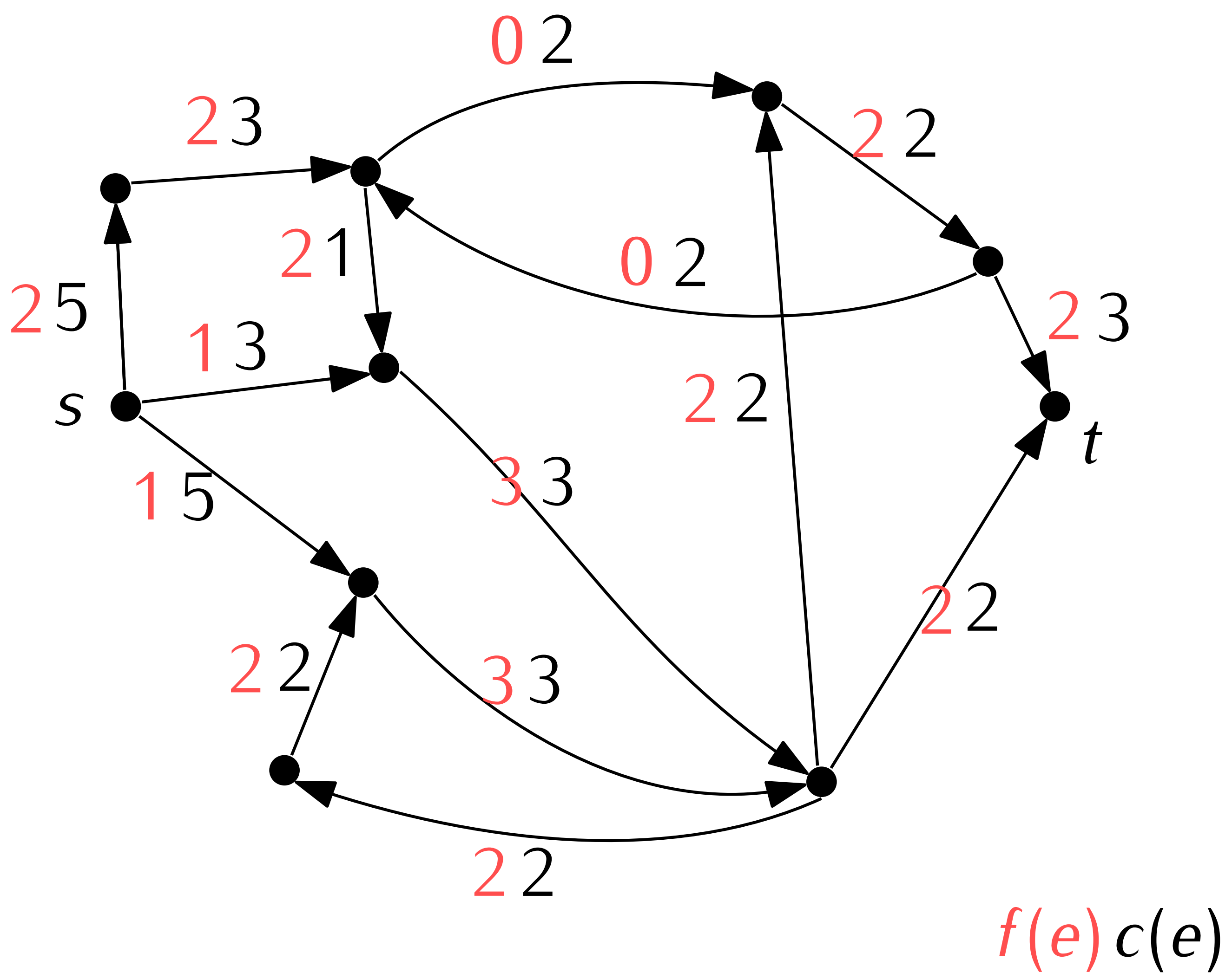
Since this holds for all s-t cuts (A, B):

Theorem. For all G, we have

$$\mathrm{maxflow}(G) \leq \min_{(A,B)\ s\text{-}t \text{ cut}} \sum_{e \text{ out of } A} c(e)$$

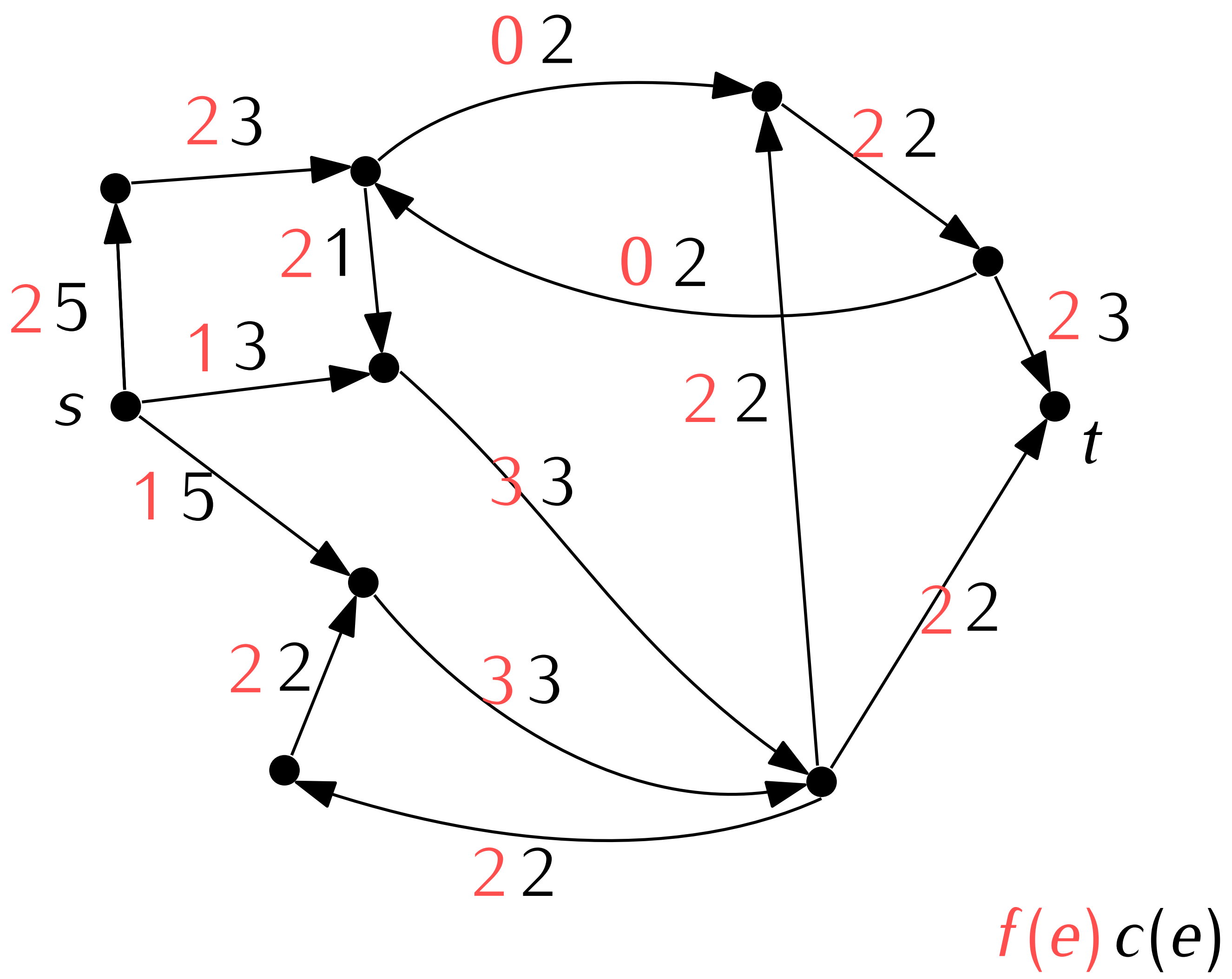


How can we know that a flow f is maximal?



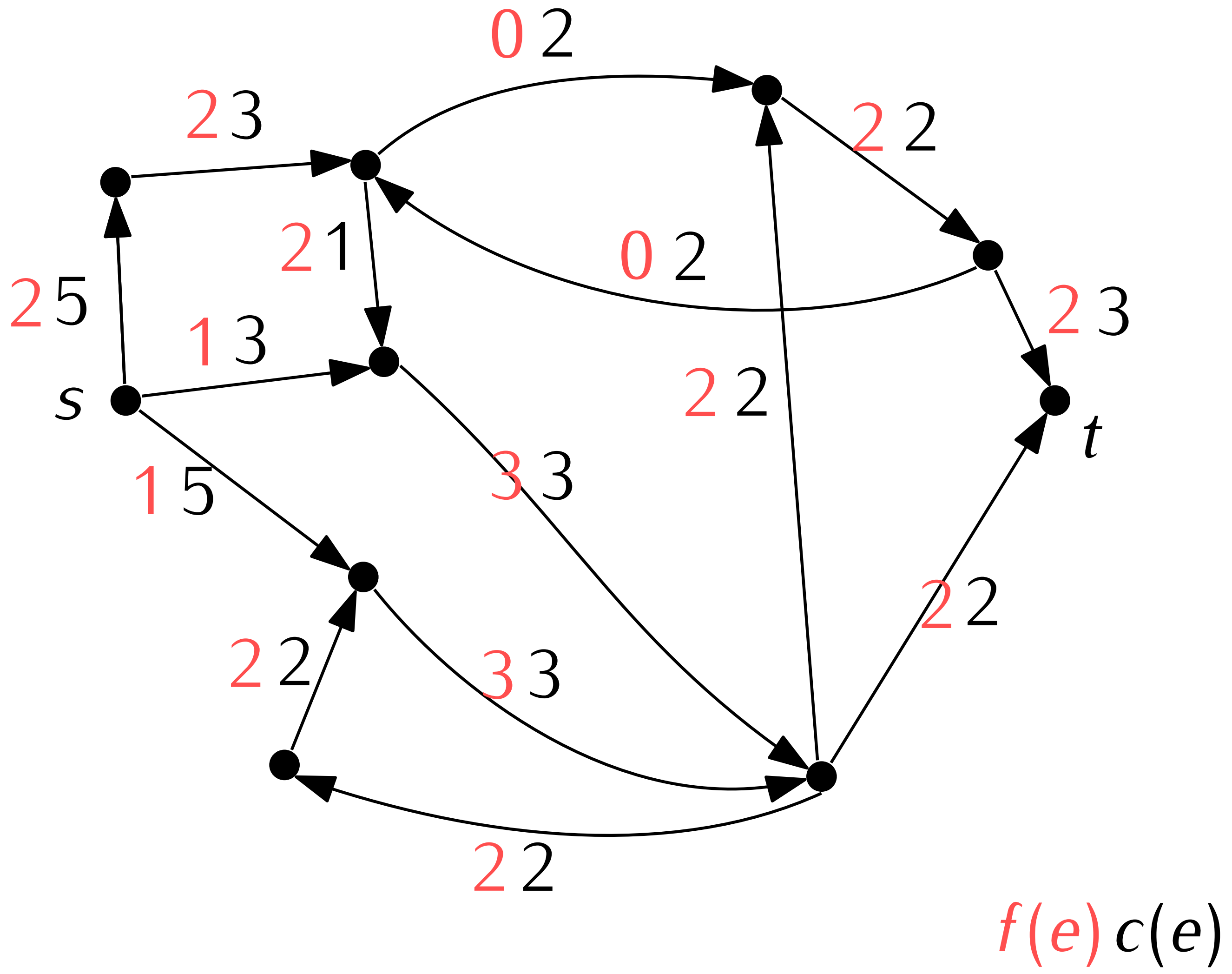
How can we know that a flow f is maximal?

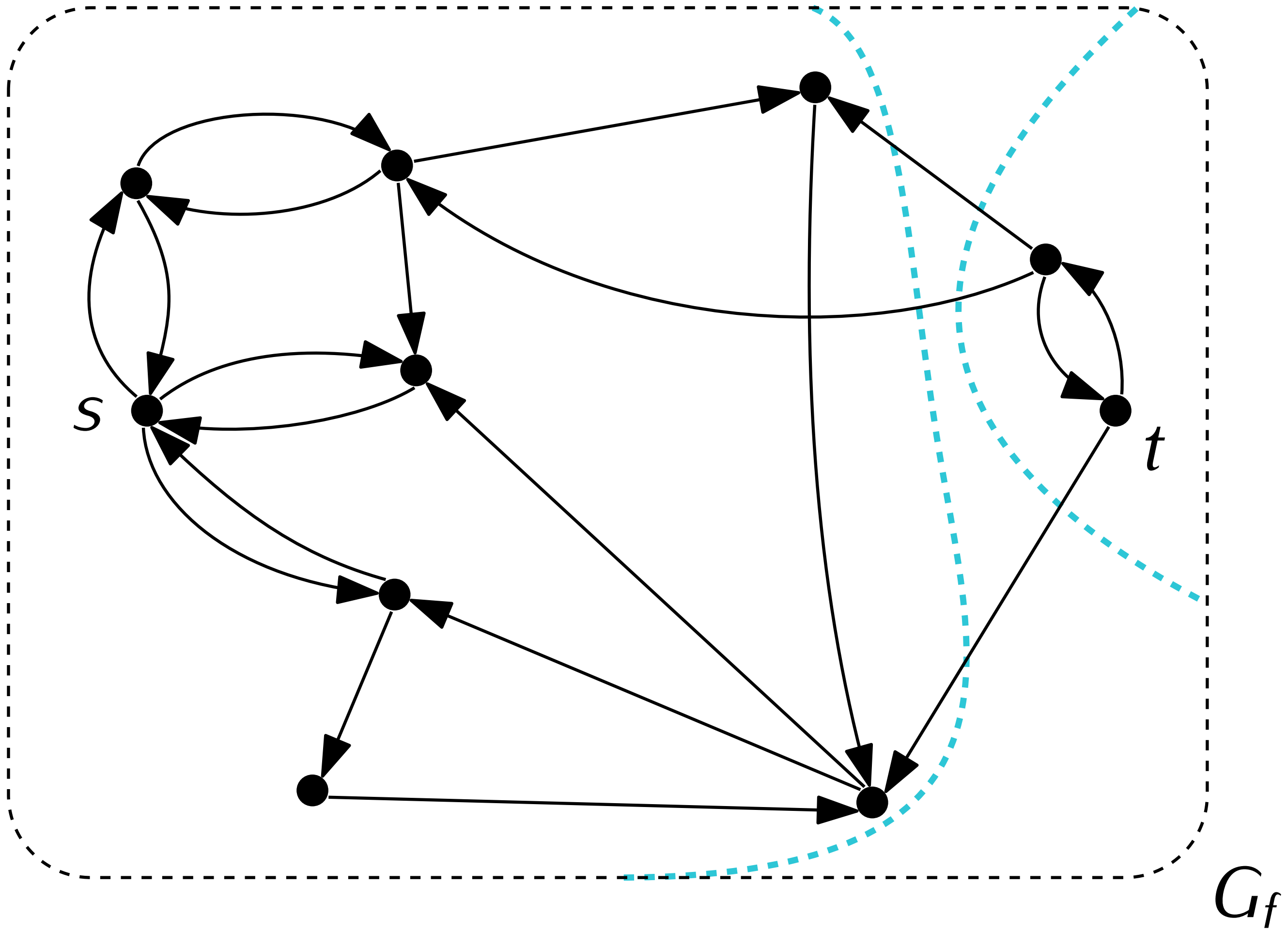
- It is enough to give a cut (A, B) such that $\sum_{e \text{ out of } A} c(e) \leq$ value of f



How can we know that a flow f is maximal?

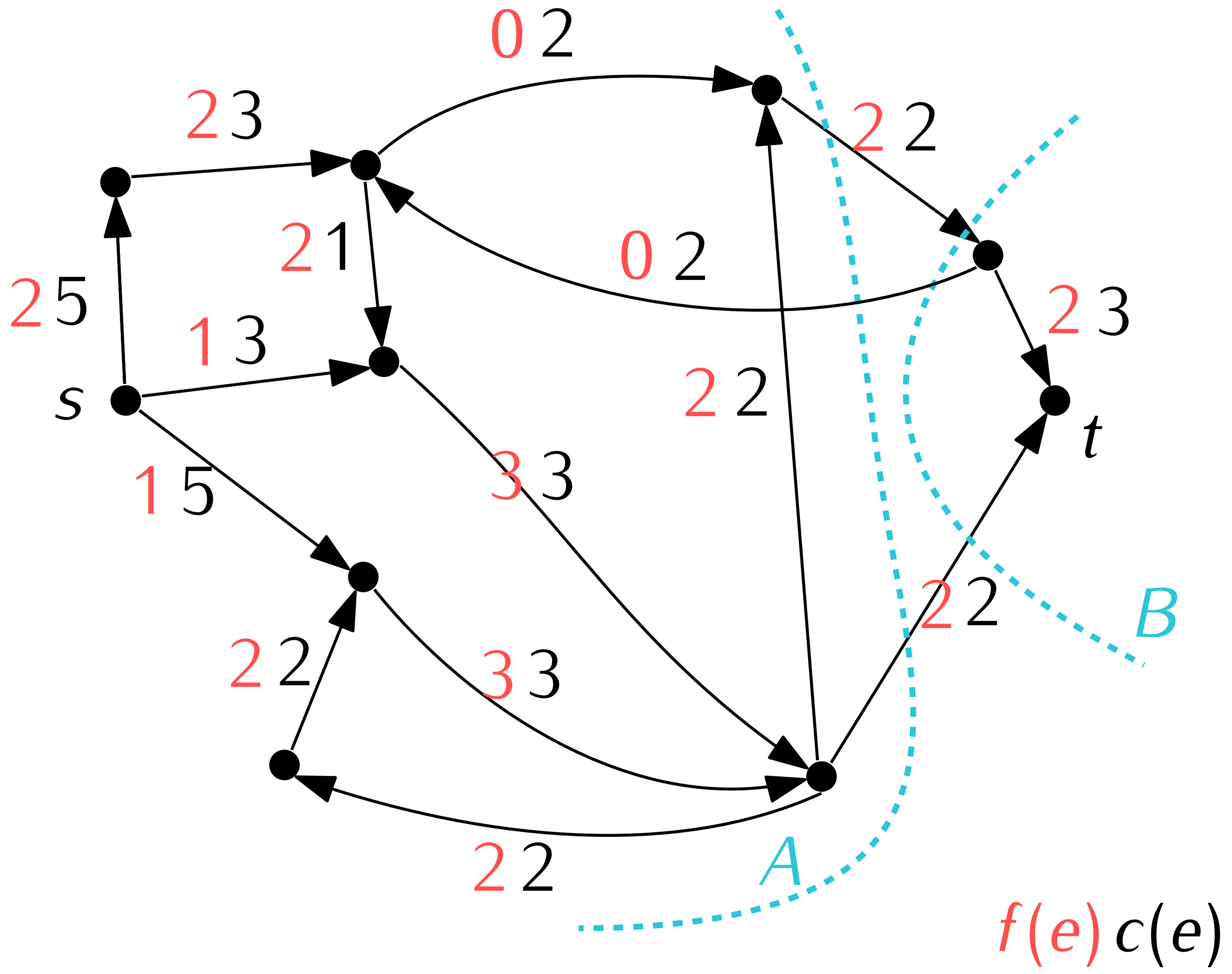
- It is enough to give a cut (A, B) such that $\sum_{e \text{ out of } A} c(e) \leq$ value of f
- $A :=$ vertices reachable from s in G_f, $B := V \setminus A$
- Since no s-t path in G_f, we have $t \in B$

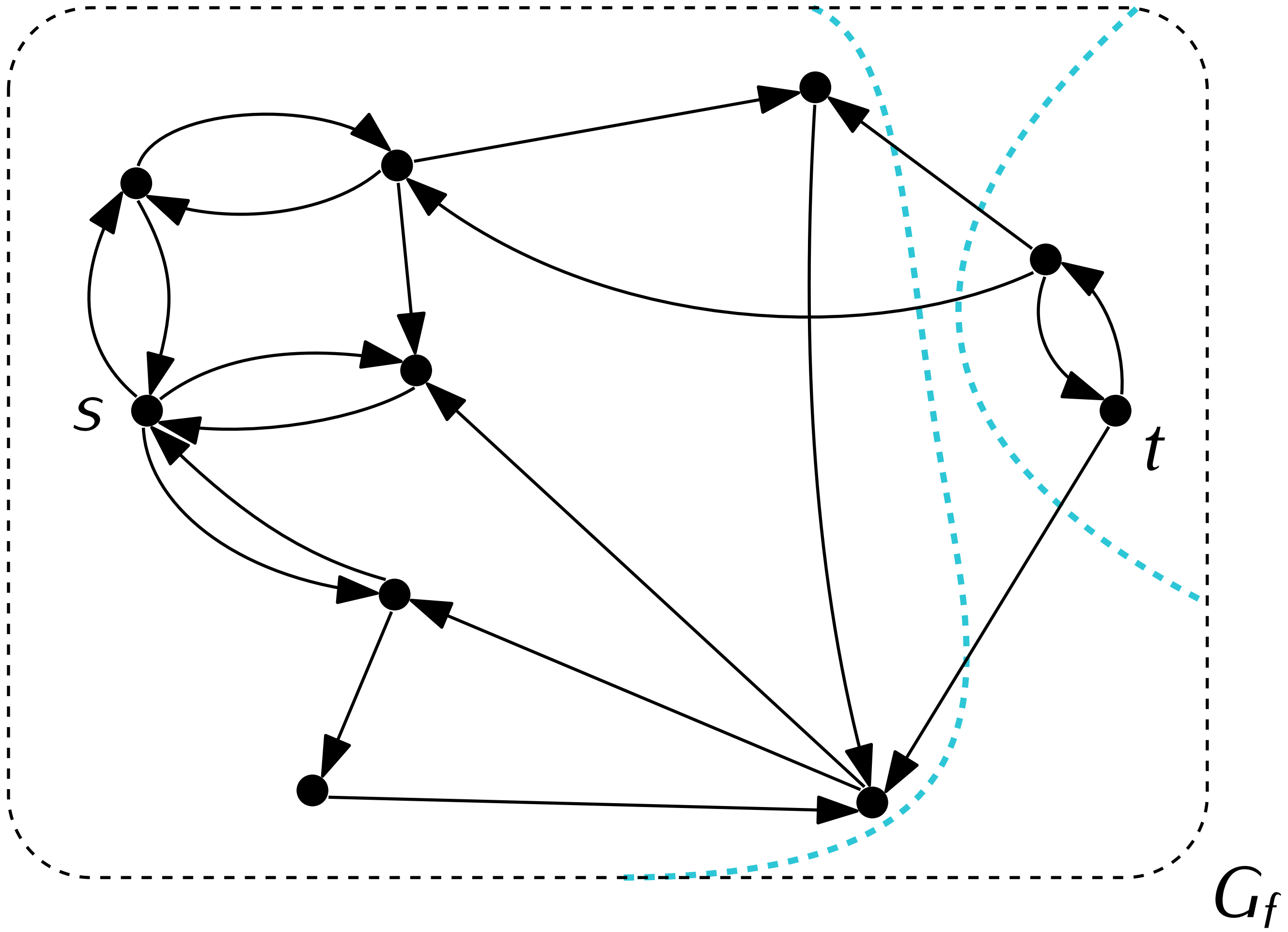




How can we know that a flow f is maximal?

- It is enough to give a cut (A, B) such that $\sum_{e \text{ out of } A} c(e) \leq$ value of f
- $A :=$ vertices reachable from s in G_f, $B := V \setminus A$
- Since no s-t path in G_f, we have $t \in B$

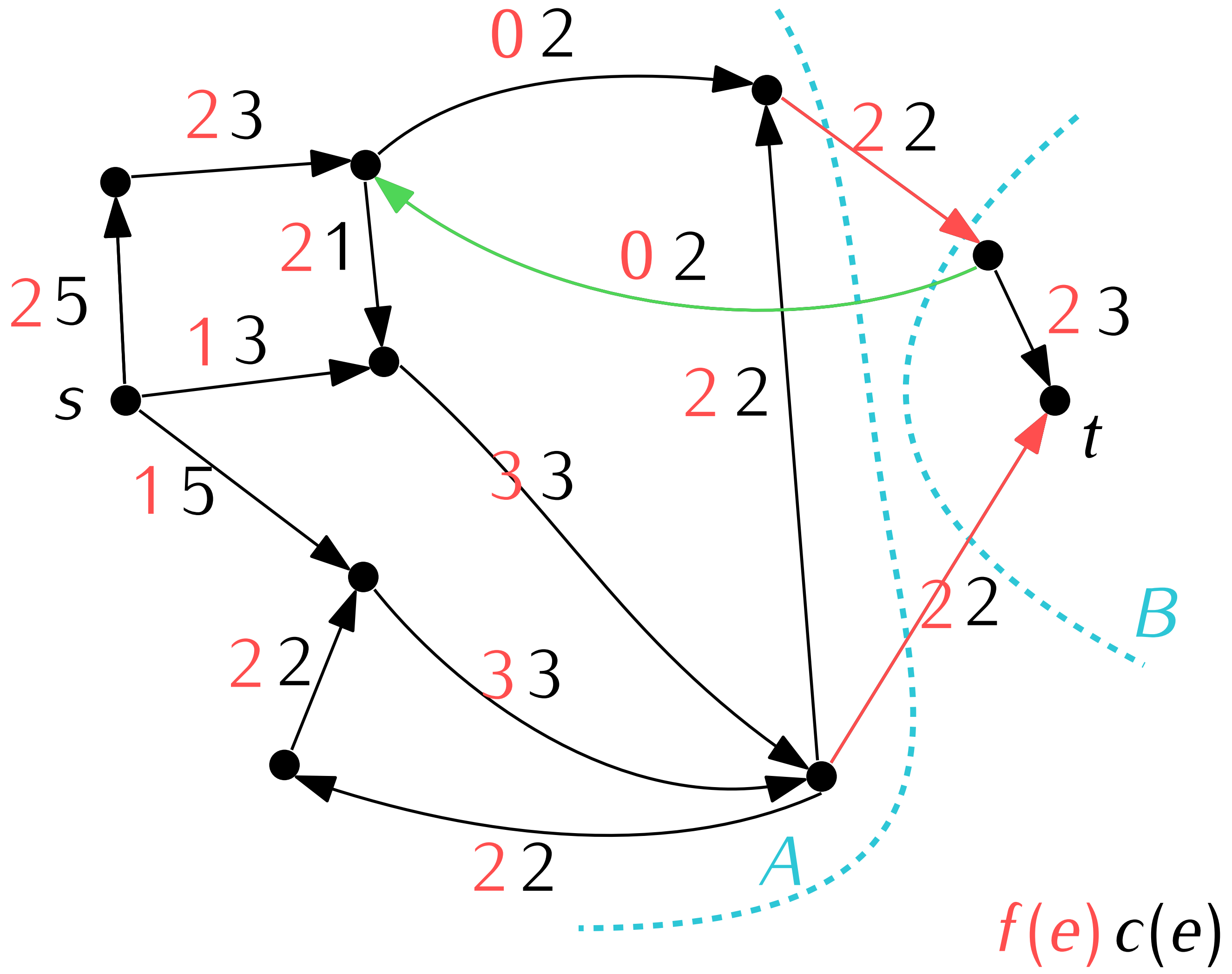


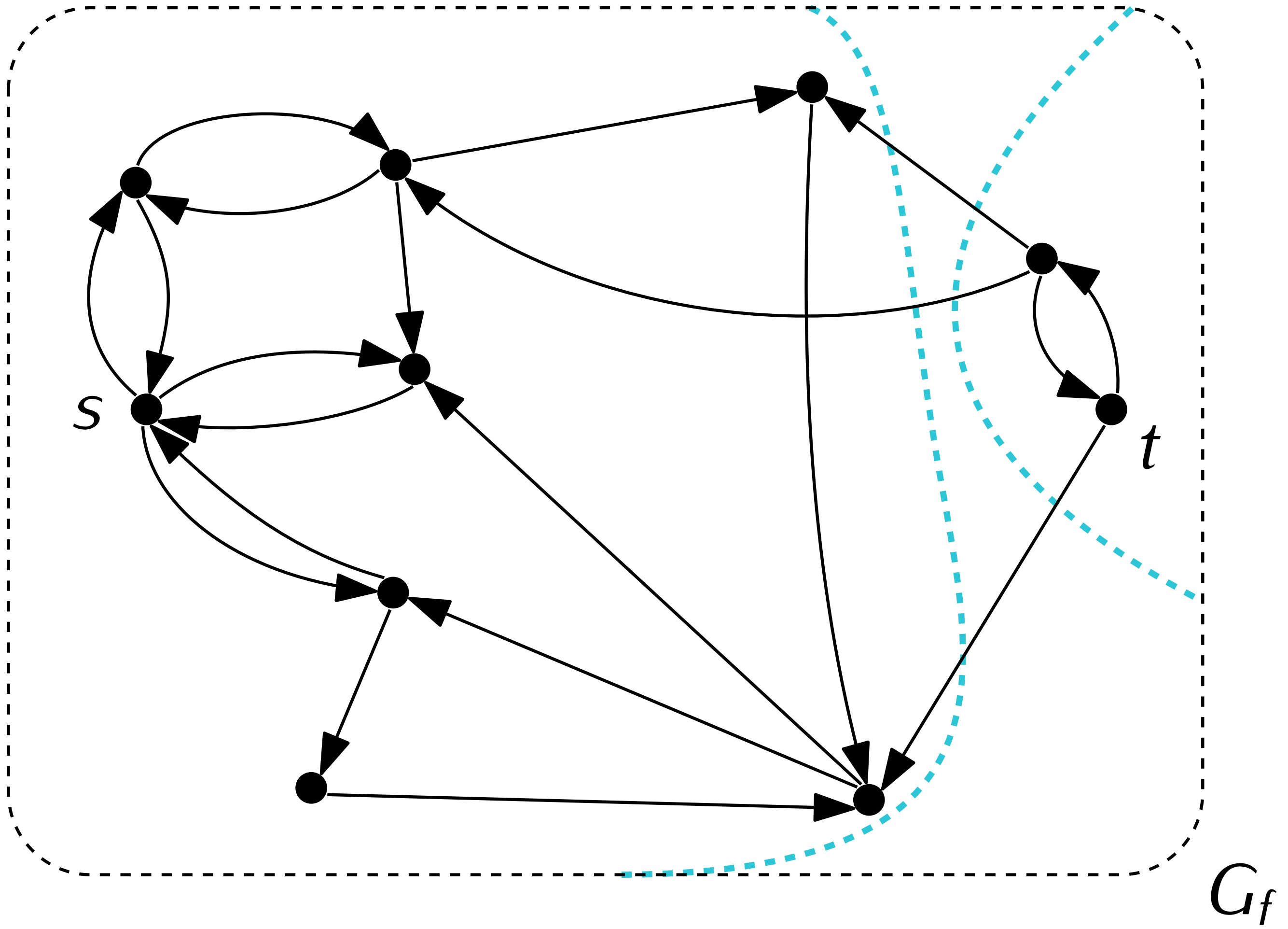


How can we know that a flow f is maximal?

- It is enough to give a cut (A, B) such that $\sum_{e \text{ out of } A} c(e) \leq$ value of f
- $A :=$ vertices reachable from s in G_f, $B := V \setminus A$
- Since no s-t path in G_f, we have $t \in B$

Lemma. All crossing edges from A to B are saturated, all crossing edges from B to A have 0 flow.





How can we know that a flow f is maximal?

- It is enough to give a cut (A, B) such that $\sum_{e \text{ out of } A} c(e) \leq$ value of f
- $A :=$ vertices reachable from s in G_f, $B := V \setminus A$
- Since no s-t path in G_f, we have $t \in B$

Lemma. All crossing edges from A to B are saturated, all crossing edges from B to A have 0 flow.

We get: value of $f = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \sum_{e \text{ out of } A} c(e) \geq \text{maxflow}(G)$

Our algorithm in fact proves an important result:

Theorem. Every flow network has a maximal flow f such that $f(e) \in \mathbb{N}$ for all $e \in E$.

Interesting applications of the algorithm for max-flow and the integrality theorem:

- Efficient algorithm for maximum bipartite matching
- Hall's perfect matching theorem
- Efficient algorithm for finding edge-disjoint paths and Menger's theorem

Our algorithm in fact proves an important result:

Theorem. Every flow network has a maximal flow f such that $f(e) \in \mathbb{N}$ for all $e \in E$.

Interesting applications of the algorithm for max-flow and the integrality theorem:

- Efficient algorithm for maximum bipartite matching
- Hall's perfect matching theorem
- Efficient algorithm for finding edge-disjoint paths and Menger's theorem

Other algorithms are possible:

$$\text{maximize } \sum_{e \text{ out of } s} c(e)$$
$$\text{subject to } \left\{ \begin{array}{rcll} \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) & = & 0 & \text{for all } v \in V \\ c(e) - f(e) & \geq & 0 & \text{for all } e \in E \\ f(e) & \geq & 0 & \text{for all } e \in E \end{array} \right.$$

Such linear programs can be solved in polynomial time: area of linear and convex optimization