CS 545

Flow Networks

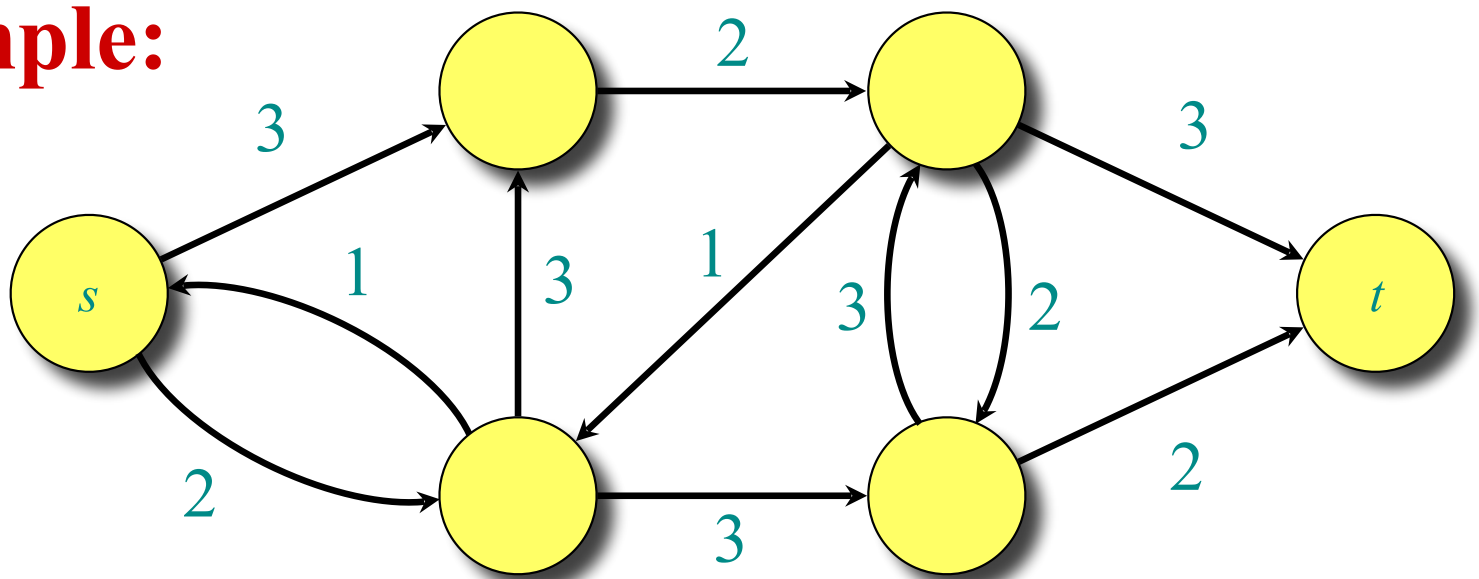
Alon Efrat

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

Flow networks

Definition. A ***flow network*** is a directed graph $G = (V, E)$ with two distinguished vertices: a ***source*** s and a ***sink*** t. Each edge $(u, v) \in E$ has a nonnegative ***capacity*** $c(u, v)$. If $(u, v) \notin E$, then $c(u, v) = 0$.

Example:



Flow networks

Definition. A ***positive flow*** on G is a function $p : V \times V \to \mathbb{R}$ satisfying the following:

- ***Capacity constraint:*** For all $u, v \in V$,

$$0 \le p(u, v) \le c(u, v).$$

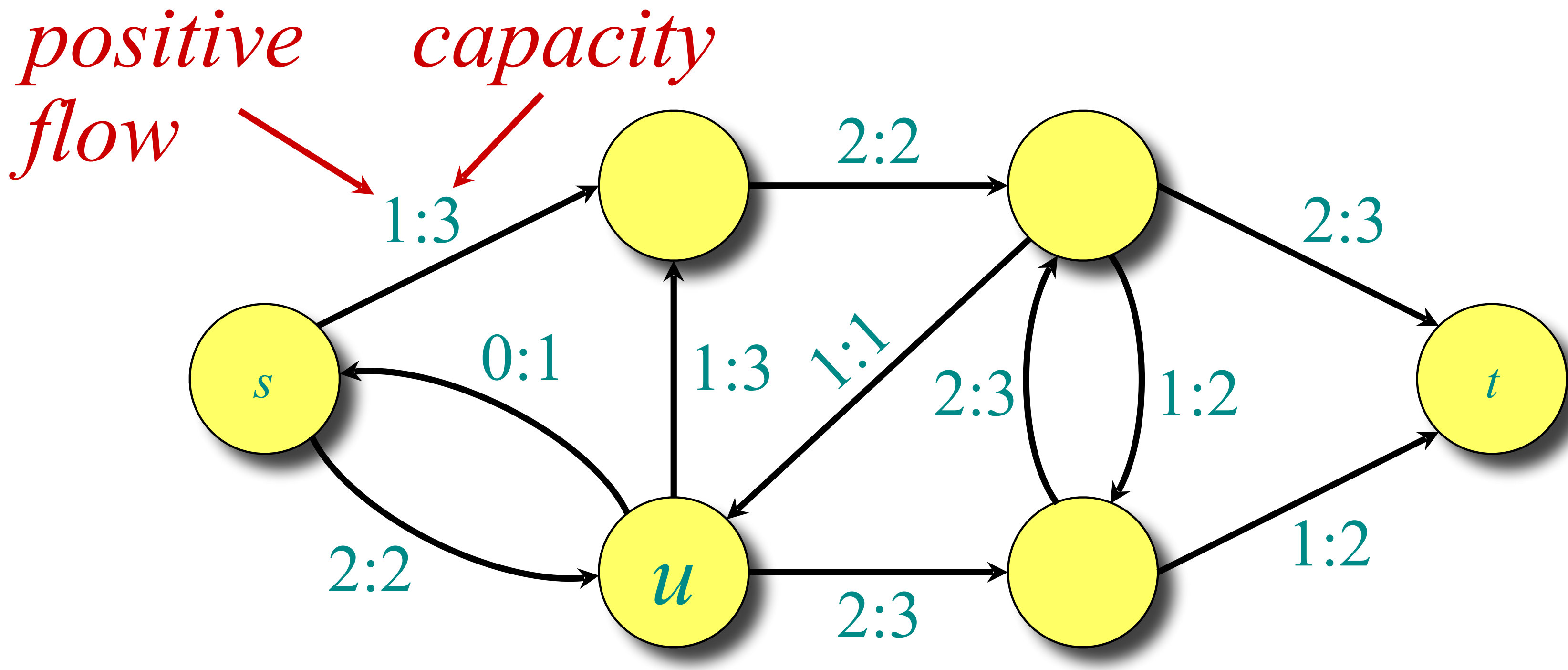
- ***Flow conservation:*** For all $u \in V - \{s, t\}$,

$$\sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u) = 0 .$$

The ***value*** of a flow is the net flow out of the source:

$$\sum_{v \in V} p(s,v) - \sum_{v \in V} p(v,s) .$$

A flow on a network



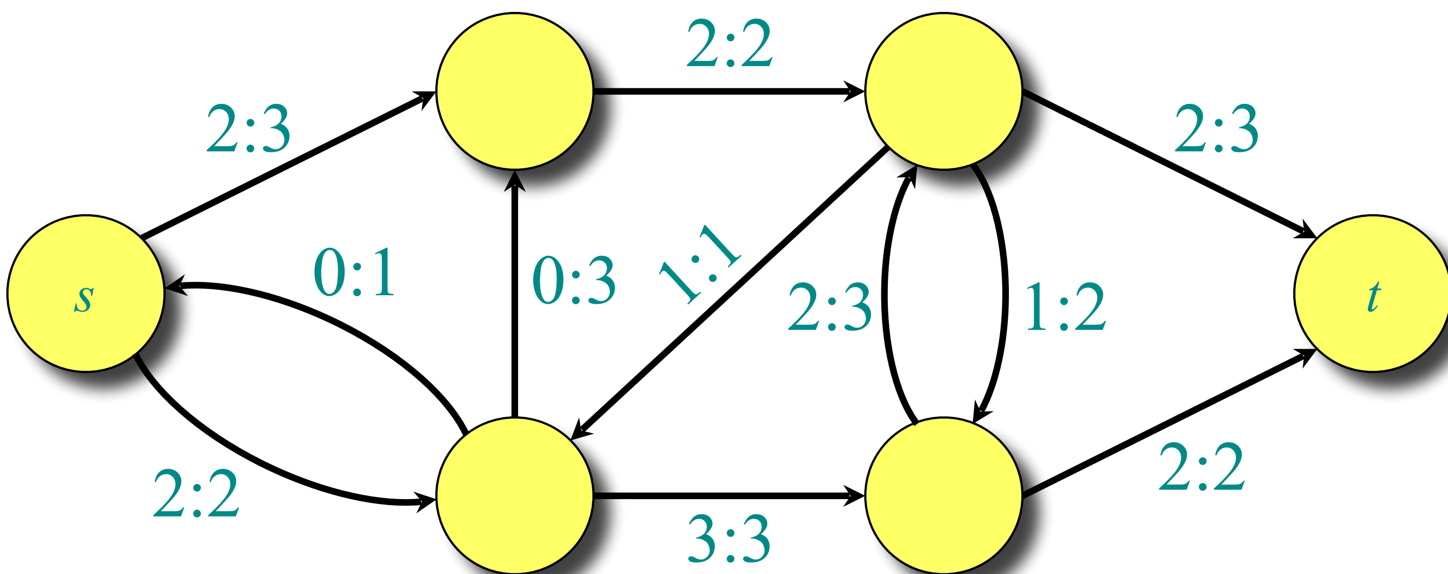
Flow conservation

- Flow into u is $2 + 1 = 3$.
- Flow out of u is $0 + 1 + 2 = 3$.

The value of this flow is $1 - 0 + 2 = 3$.

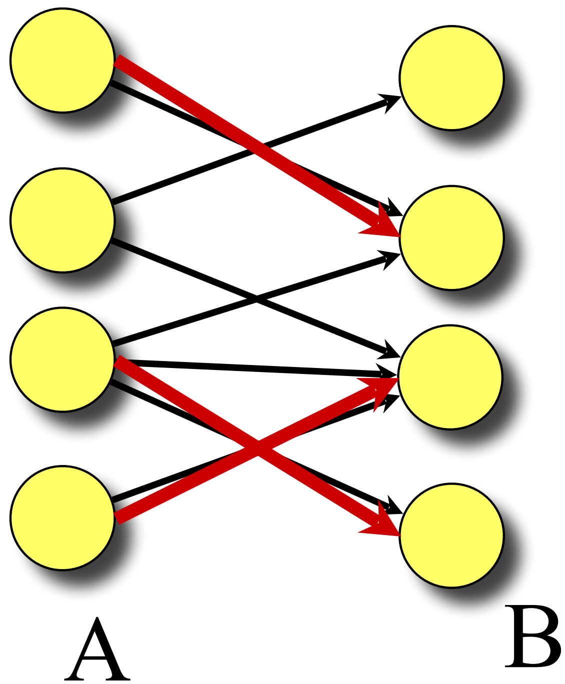
The maximum-flow problem

Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



The value of the maximum flow is 4.

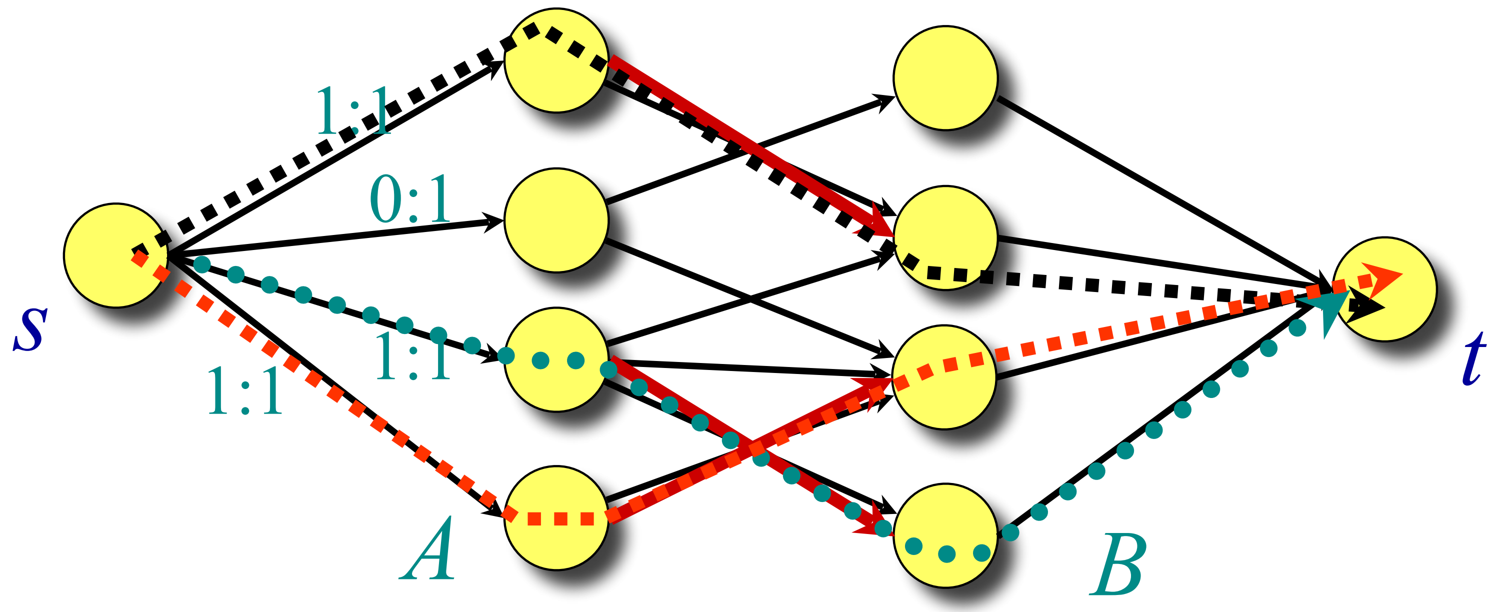
Application: Bipartite Matching.



A graph *G(V,E)* is called **bipartite** if *V* can be partitioned into two sets $V = A \cup B$, and each edge of *E* connects a vertex of *A* to a vertex of *B*.

A **matching** is a set of edges *M* of *E*, where each vertex of *A* is adjacent to at most one vertex of *B, and vice versa.*

Matching and flow problem



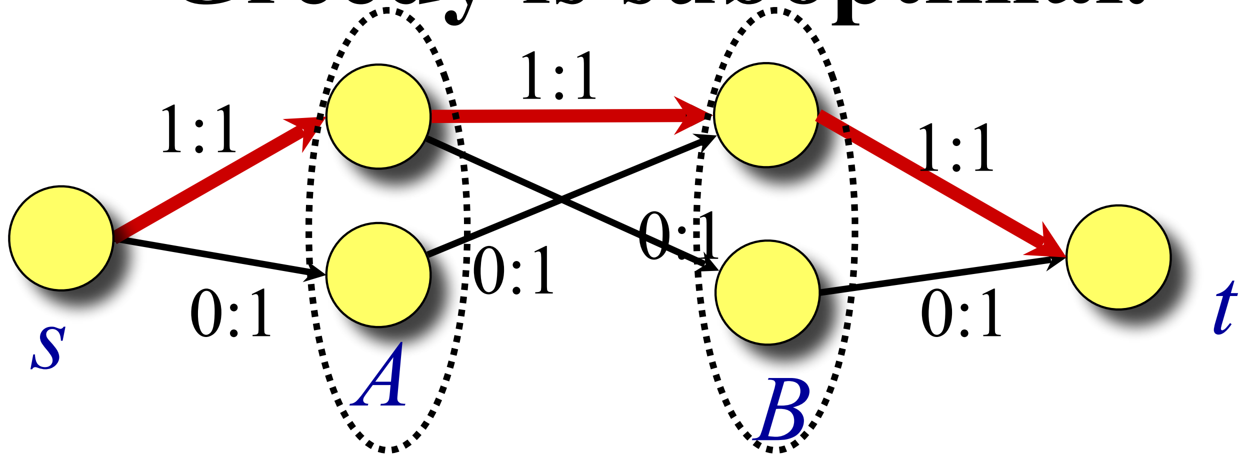
Add a vertex *s*, and connect it to each vertex of *A*.
Add a vertex *t*, and connect each vertex of *B* to *t*.
The capacity of all edges is 1.

Find max flow. Assume it is an **integer** flow, so the flow of each edge is either 0 or 1.

Each edge of *G* that carries flow is in the matching.
Each edge of *G* that **does not** carry flow is **not in** the matching.

Claim: The edge between *A* and *B* that carry flow, form a matching.

Greedy is suboptimal.



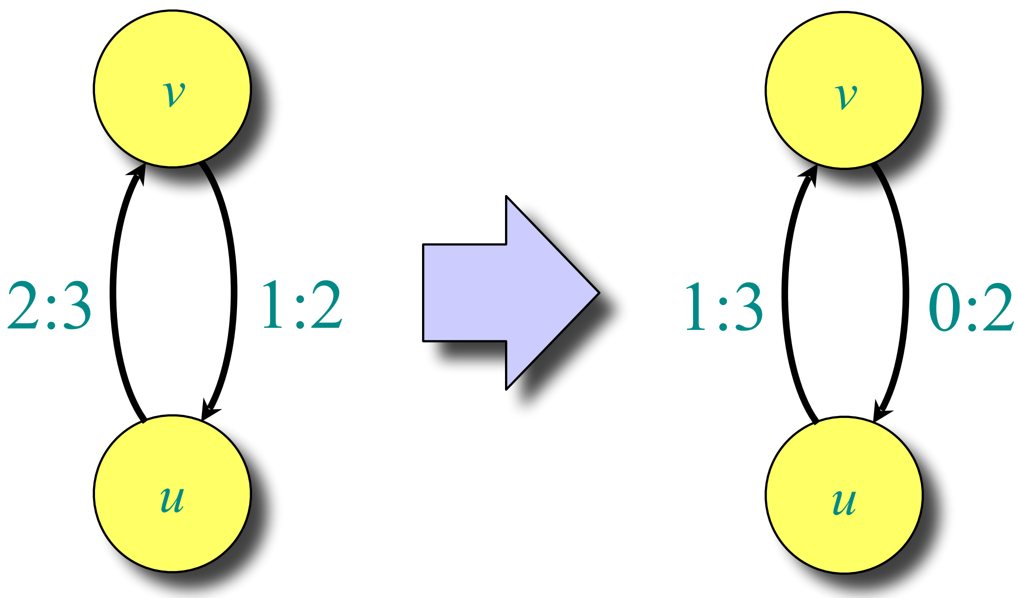
Assume we have already some edges in a (partial) matching M.

In order to increase the cardinality of the matching we might need to first remove from M some edges (somehow counterintuitive ?)

Thinking again about the matching as flow problem, it means that we might need to remove flow from edges that currently curry flow.

Flow cancellation

Without loss of generality, positive flow goes either from *u* to *v*, or from *v* to *u*, but not both.



Net flow from *u* to *v* in both cases is 1.

The capacity constraint and flow conservation are preserved by this transformation.

A notational simplification

IDEA: Work with the net flow between two vertices, rather than with the positive flow.

Definition. A ***(net) flow*** on G is a function $f : V \times V \to \mathbb{R}$ satisfying the following:

- ***Capacity constraint:*** For all $u, v \in V$,

$$f(u, v) \le c(u, v).$$

- ***Flow conservation:*** For all $u \in V - \{s, t\}$,

$$\sum_{v \in V} f(u, v) = 0.$$

← *One summation instead of two.*

- ***Skew symmetry:*** For all $u, v \in V$,

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

Equivalence of definitions

Net flow *vs*. positive flow.

Theorem. The two definitions are equivalent.

Proof. (from positive flow to net flow)

Let $f(u, v) = p(u, v) - p(v, u)$.

- ***Capacity constraint:*** Since $p(u, v) \le c(u, v)$ and $p(v, u) \ge 0$, we have $f(u, v) \le c(u, v)$.
- ***Flow conservation:***

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} \left(p(u,v) - p(v,u) \right) = \sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u)$$

- In particular, if $u \in V - \{s, t\}$, then $\sum_{v \in V} f(u,v) = 0$

- ***Skew symmetry:***

$$\begin{aligned} f(u, v) &= p(u, v) - p(v, u) \\ &= -(p(v, u) - p(u, v)) \\ &= -f(v, u). \end{aligned}$$

The two definitions (net vs positive) are equivalent

Second direction: Assume net $f(u, v)$ is given, generate legit positive $p(u, v)$ Define

$$p(u, v) = \begin{cases} f(u, v) & \text{if } f(u, v) > 0, \\ 0 & \text{if } f(u, v) \le 0. \end{cases}$$

- ***Capacity constraint:*** By definition, $p(u, v) \ge 0$. Since $f(u, v) \le c(u, v)$, it follows that $p(u, v) \le c(u, v)$.
- ***Flow conservation:*** If $f(u, v) > 0$, then $f(v, u) < 0$ so $p(v,u)=0$.

  $$p(u, v) - p(v, u) = f(u, v).$$

  If $f(u, v) \le 0$, then

  $$p(u, v) - p(v, u) = -f(v, u) = f(u, v)$$ by skew symmetry.

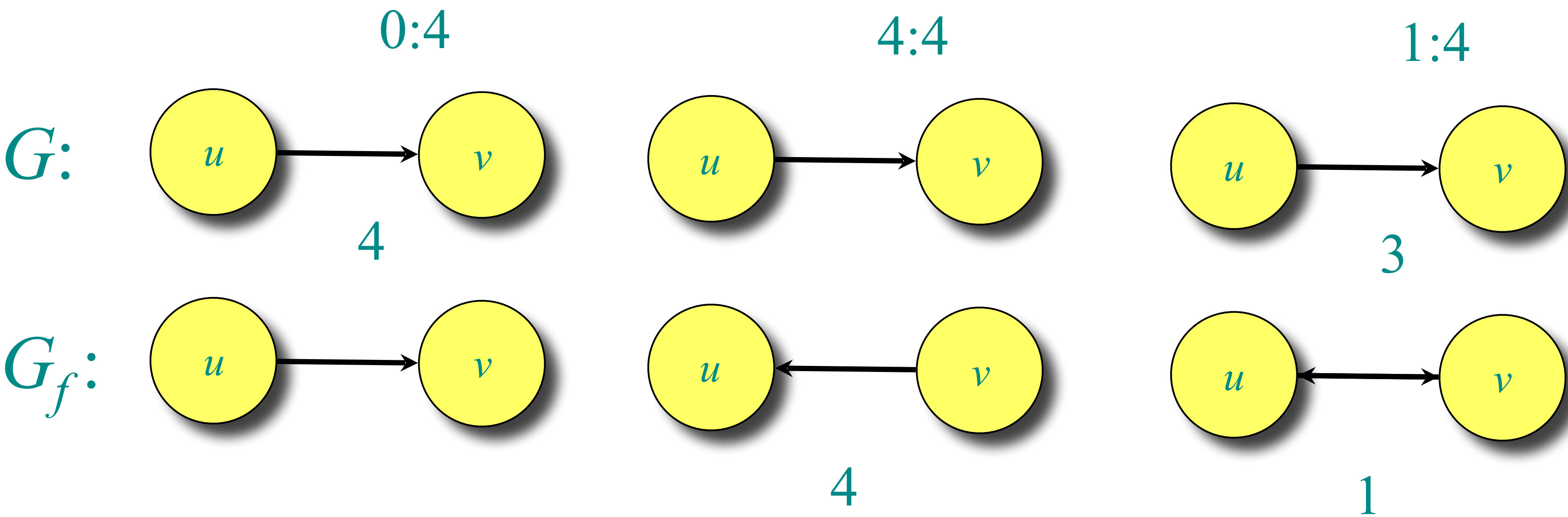
  Therefore,

$$\sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u) = \sum_{v \in V} f(u,v)$$

Residual network

Definition. Let f be a flow on $G = (V, E)$.
The ***residual network*** $G_f(V, E_f)$ is the graph with strictly positive ***residual capacities*** $c_f(u, v) = c(u, v) - f(u, v) > 0$.

Examples:



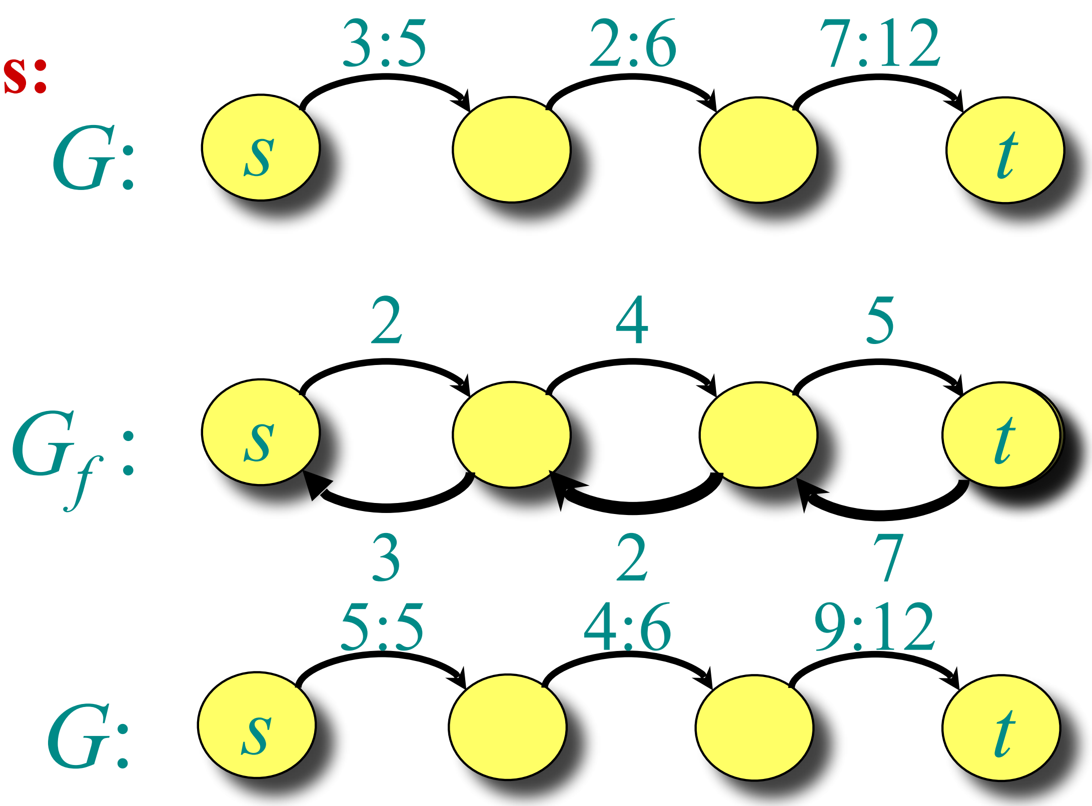
Lemma. $|E_f| \leq 2|E|$. □

Augmenting paths

- **Definition.** Any path from s to t in G_f is an ***augmenting path*** in G with respect to f.
- The flow value can be **increased** along an augmenting path p by adding
. $c_f(p) := \min\{ c_f(u,v) \mid (u,v) \in p \}$ to the net flow of each edge along p.
- $\forall\ (u,v) \in p$ set $f(u,v) \mathrel{+}= c_f(p)$; $f(v,u) \mathrel{-}= c_f(p)$
- This is called **path augmentation**.

Examples:

$c_f(p) = 2$



Note – flow conservation is preserved.

Augmenting paths – another example

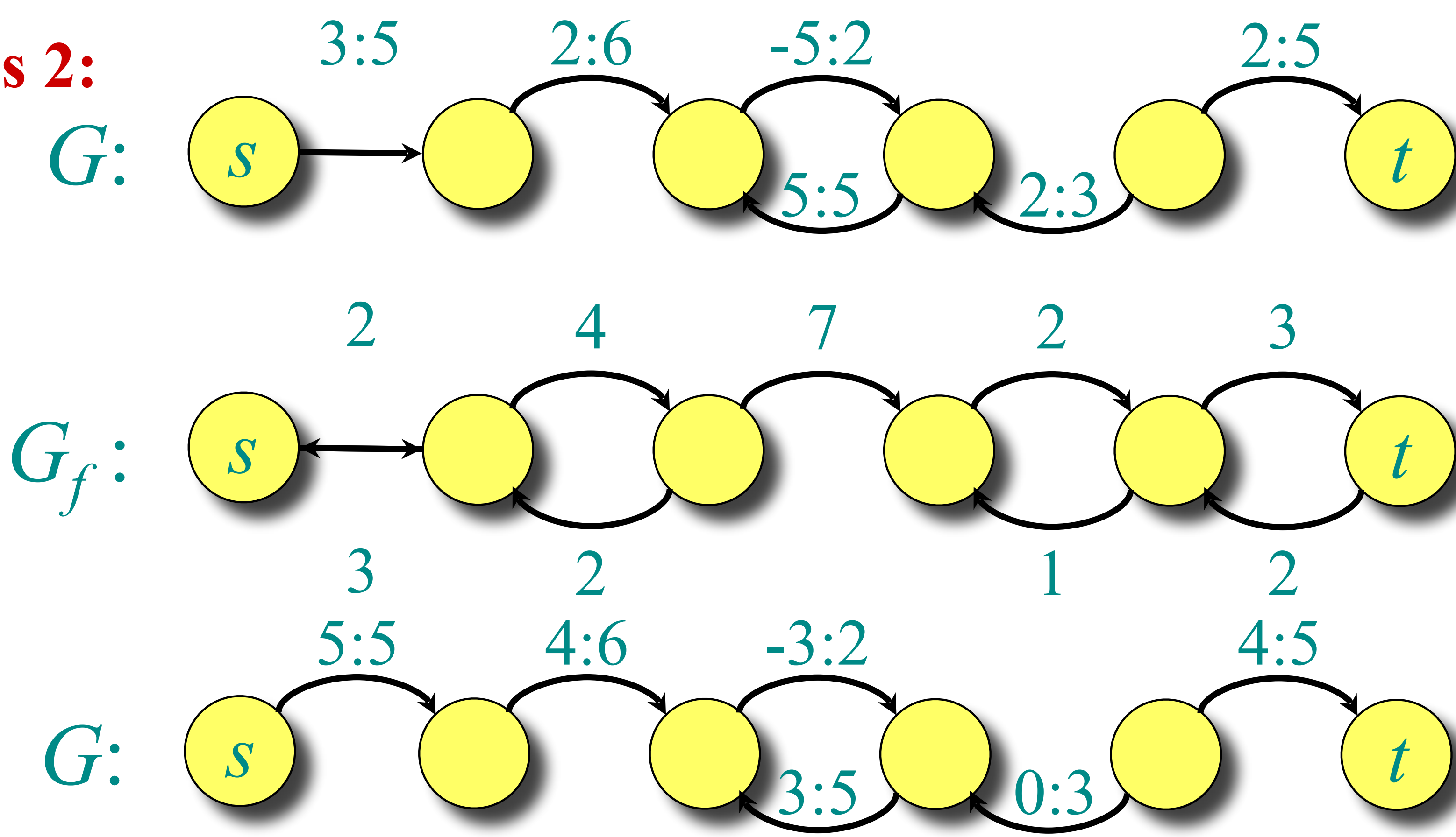
•**Definition.** Any path from s to t in G_f is an ***augmenting path*** in G with respect to f.

•The flow value can be **increased** along an augmenting path p by adding $c_f(p) := \min\{ c_f(u,v) \mid (u,v) \text{ on } p \}$ to the net flow of each edge along p.

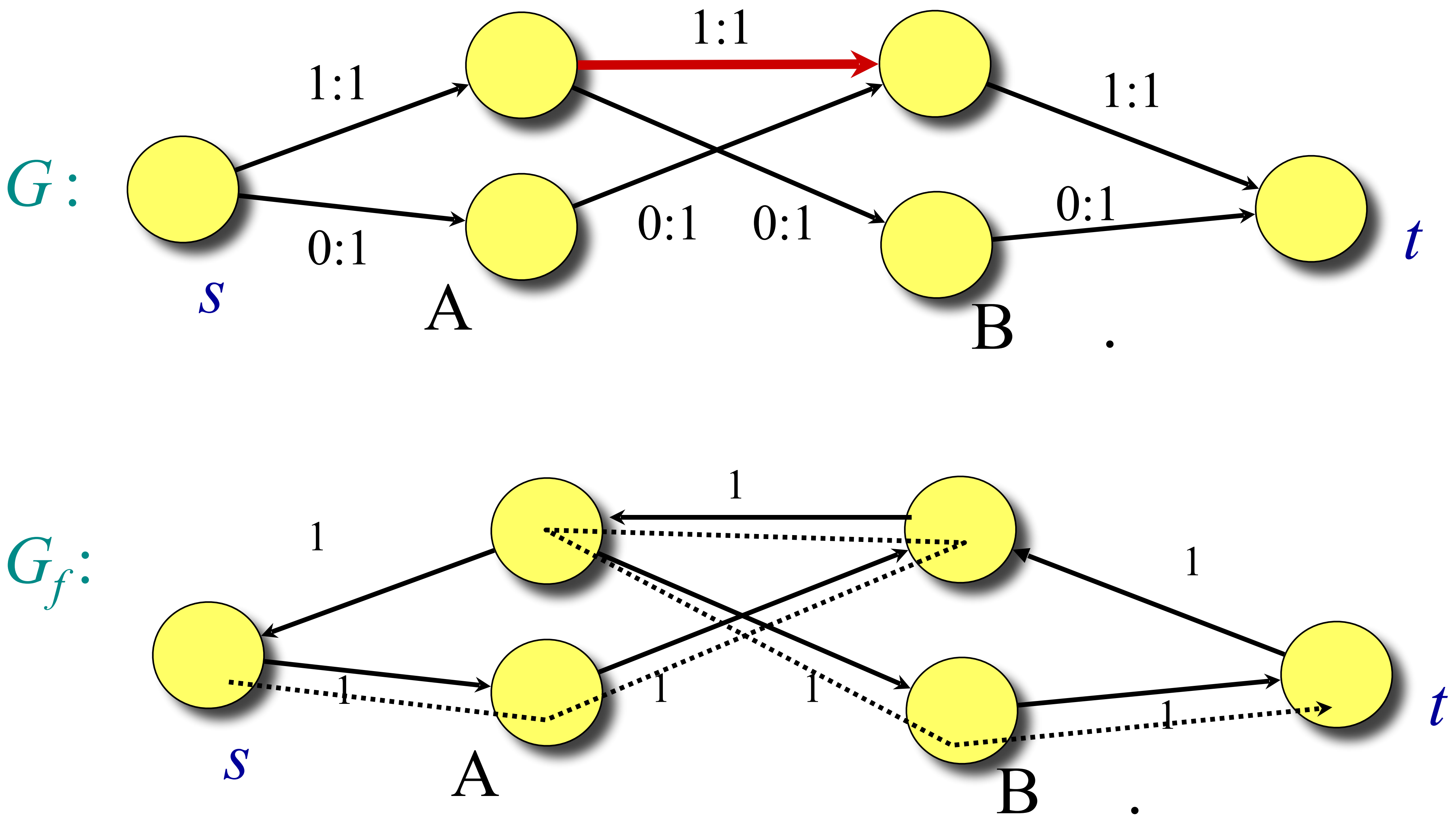
•This is called **path augmentation**.

Examples 2:

$c_f(p) = 2$

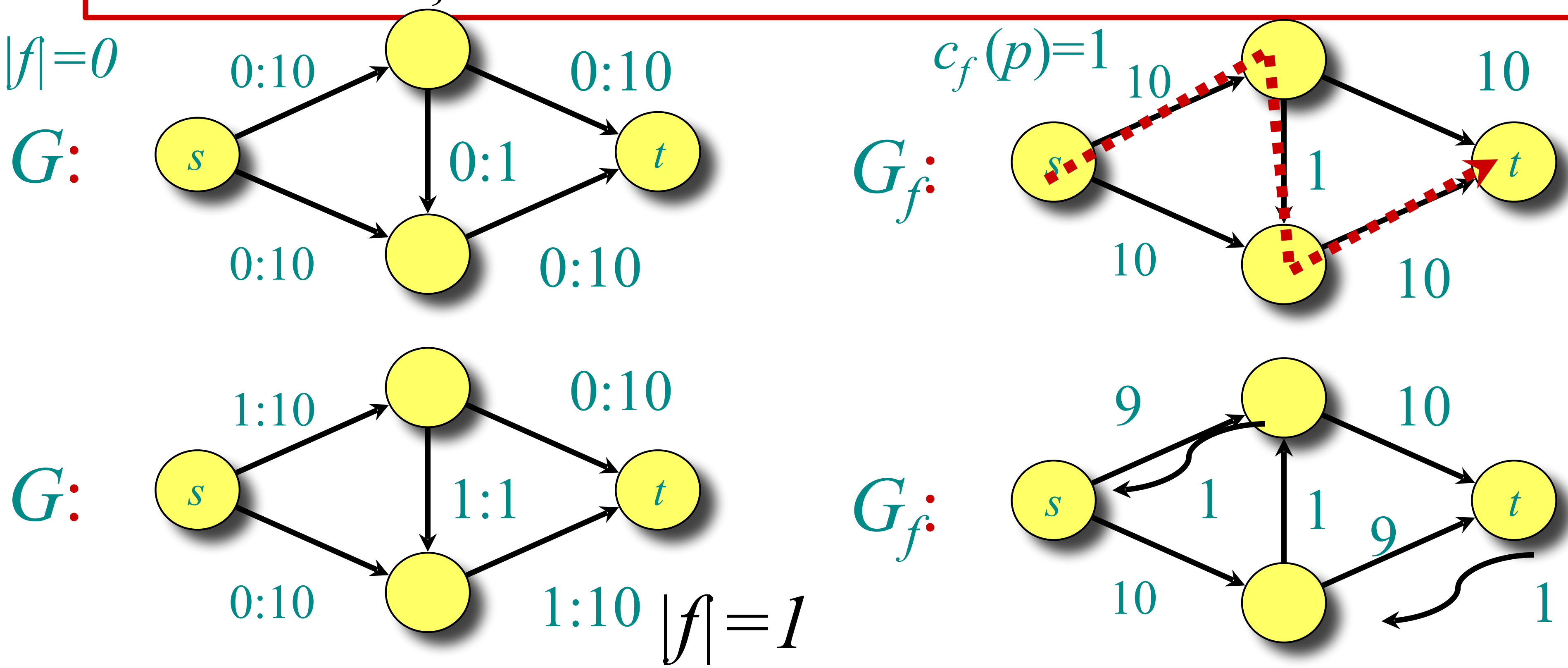


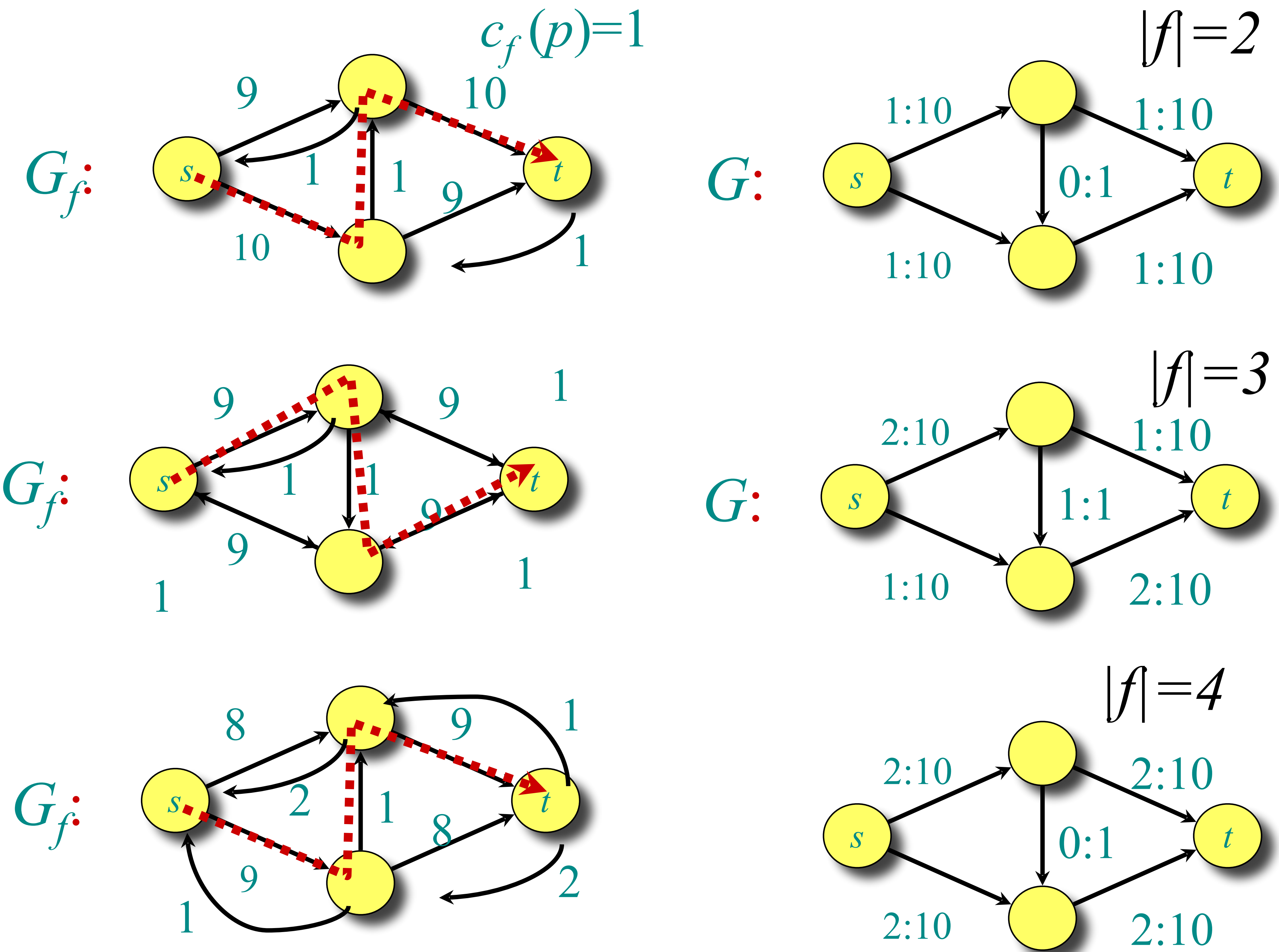
Example – maximum matching



Ford-Fulkerson max-flow algorithm

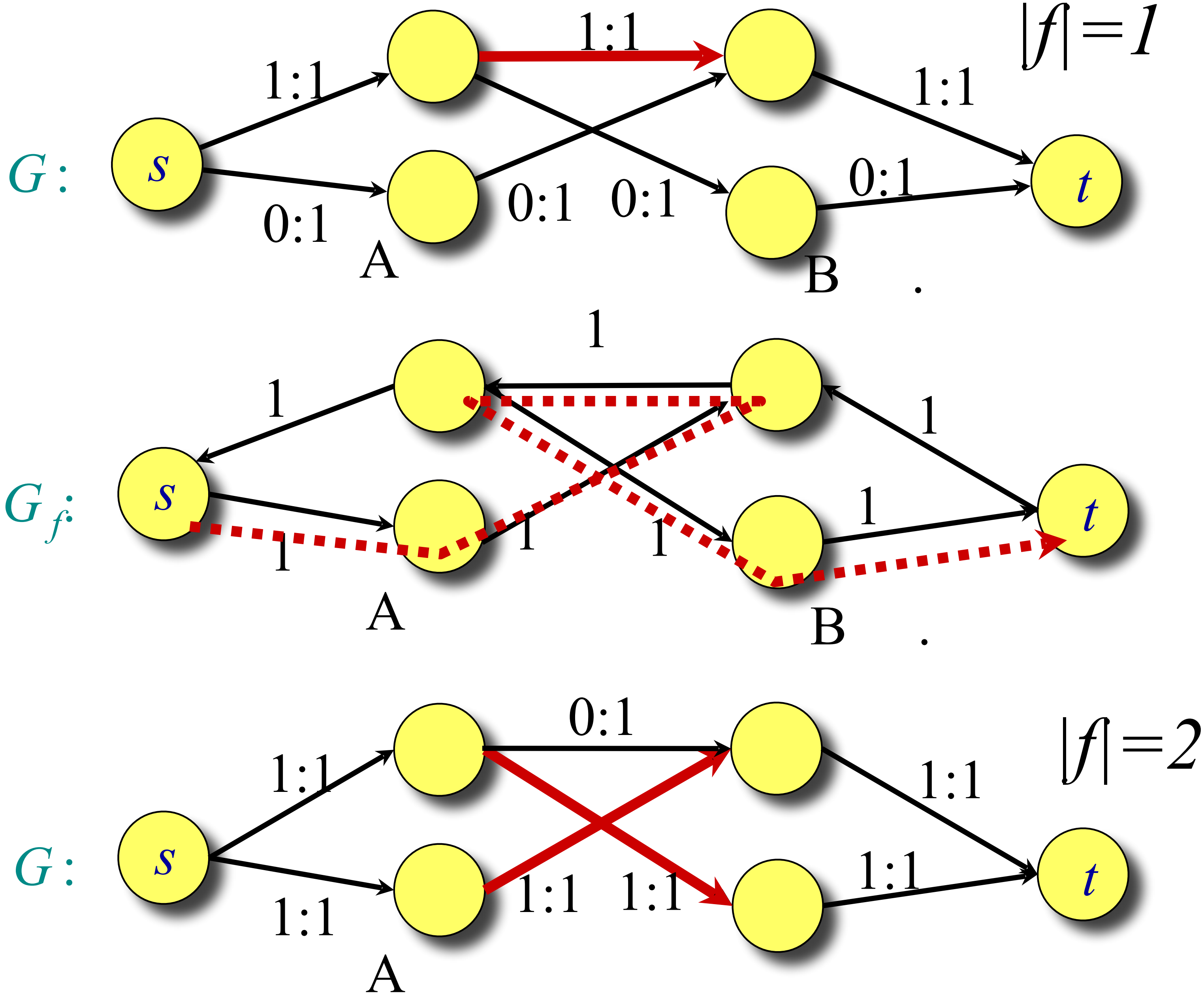
- Start: $f[u, v] \leftarrow 0$ for all $u, v \in V$
- While **(1)** {
  - construct G_f
  - if an augmenting path p in G_f exists then
    augment f by $c_f(p)$ //Any path would do
  - else exit }





$c_f(p)$=1
G_f:
s
t
9
10
1
1
9
10
1
|f|=2
G:
s
t
1:10
1:10
0:1
1:10
1:10
G_f:
s
t
9
9
1
1
1
9
9
1
1
|f|=3
G:
s
t
2:10
1:10
1:1
1:10
2:10
G_f:
s
t
8
9
1
2
1
8
9
2
1
|f|=4
s
t
2:10
2:10
0:1
2:10
2:10

Another example - Matching



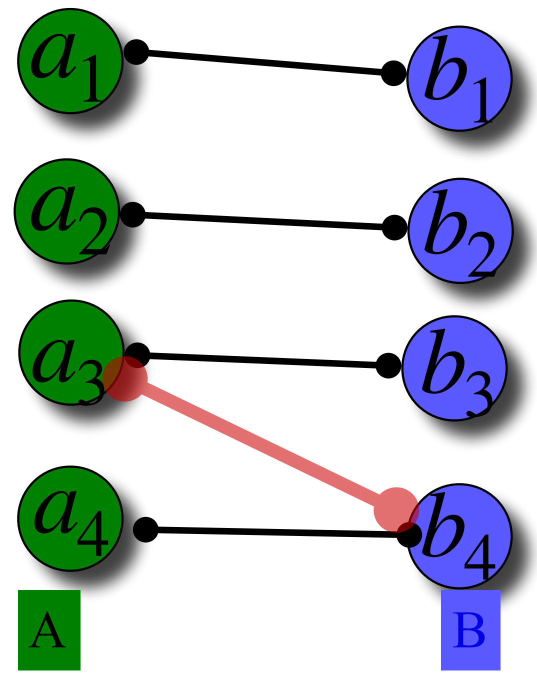
Ford-Fulkerson algorithm for finding max bipartite matching

- This algorithm is actually appropriate for any network flow problem, but notations and proofs are simpler if we concentrate on the matching directly.

- Algorithm: Start then $M = \emptyset$. No edge is in the matching.
- Output: $|M|$ is as large as possible
- At each step of the algorithm, we increase the cardinality of M by 1.

- General Step: Assume M is given. Terminology:
- A ***matched vertex*** is a vertex which is an edgepoint of an edge of M. Vertices that are not matched are called ***exposed vertices***.

- We will denote all matched edge $M \subseteq E$ by a thick red segments, and edge of $E \backslash M$ are depicted by a straight edge. Sometimes we will denote these edges by 

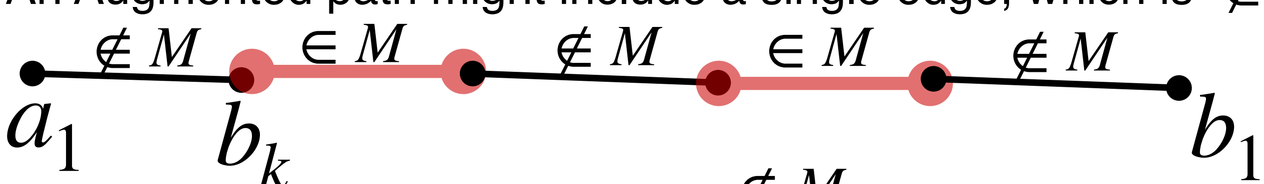


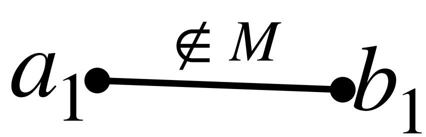
A **matching** is a set of edges M of E, where each vertex of A is adjacent to at most one vertex of B, and vice versa.

In red: Edge of the matching

- An **augmenting path** is a path that starts with an expose vertex of A, ends at an exposed vertex of B, and its edges alternates: An edge $\notin M$ followed by an edge $\in M$, followed by an edge $\notin M$ and so on.

- An Augmented path might include a single edge, which is $\notin M$





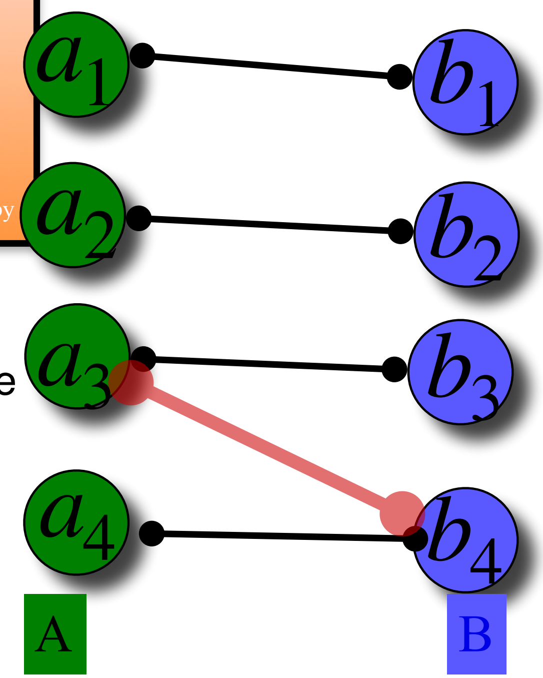
Ford-Fulkerson algorithm for finding max bipartite matching

This algorithm is actually appropriate for any network flow problem, but notations and proofs are simpler if we concentrate on the matching directly.

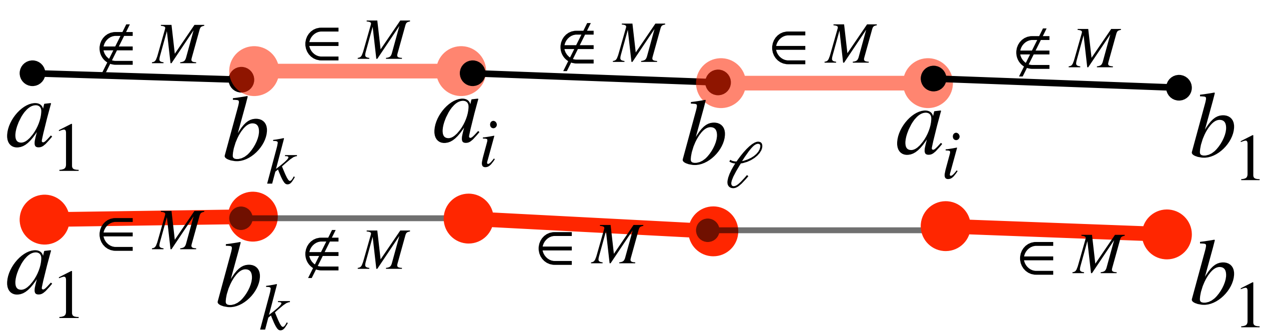
Algorithm: Start then $M = \emptyset$. No edge is in the matching.
Output: $|M|$ is as large as possible
At each step of the algorithm, we increase the cardinality of M by 1.

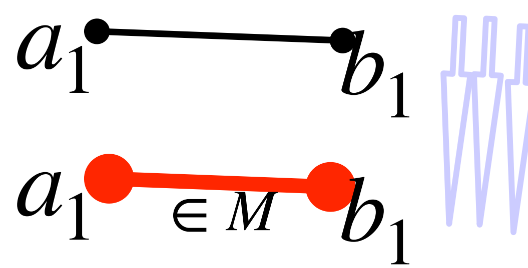
General Step: Assume M is given. Terminology:
A ***matched vertex*** is a vertex which is an edgepoint of an edge of M. Vertices that are not matched are called ***exposed vertices***.

We will denote all matched edge $M \subseteq E$ by a thick red segments, and edge of $E \setminus M$ are depicted by a straight edge. Sometimes we will denote these edgeb_ks by
An **augmenting path** is a path that starts with an expose vertex of A, ends at an exposed vertex of B, and its edges alternates: An edge $\notin M$ followed by an edge $\in M$, followed by an edge $\notin M$ and so on.



- An **augmenting path** is a path that starts with an expose vertex of A, ends at an exposed vertex of B, and its edges alternates: An edge $\notin M$ followed by an edge $\in M$, followed by an edge $\notin M$ and so on.
- Augmented path might include a single edge, which is $\notin M$
- Lets p be an augmenting path. The operation of **augmentation a path** consists of
  - Insert into M all edges of p which are $\notin M$, and remove from M all edges that originally were in M.



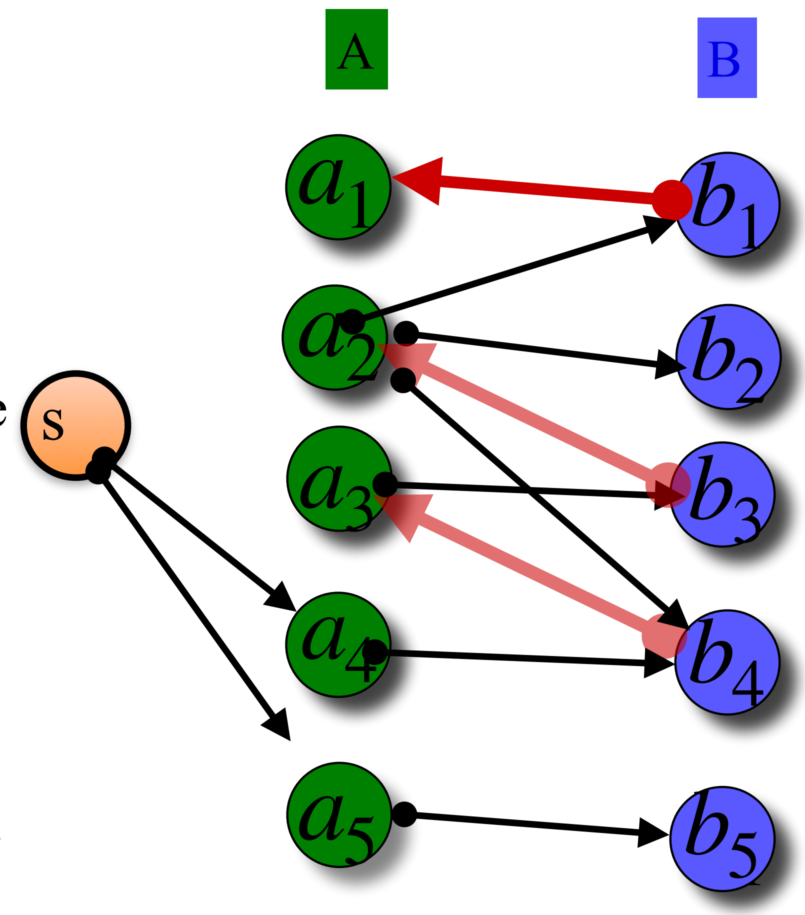


Claims:

1. A vertex that was matched before the augmentation, is matched after the augmentation
2. Matching is 1-1 (no course taught by two teaches, no teacher teaches two courses.
3. Augmentation increases the number of edges in the matching by 1.

How to find augmenting paths

- Makes the graph a directed graph:
  - Edges $\in M$ are directed from right to left
  - Edges $\notin M$ are directed from left to right
  - Add a vertex s, and connect it to every **exposed** $a_i \in A$
- Run DFS or BFS from s.
- Every path that leads to an exposed vertex must be an augmented path. And
- If there is an augmented path, this process will find this path.



Once an augmented bath is found, we augment its edges, and restart (re-bulding the directed graph).

If no augmented path is found, stop - M is maximum cardinality matching. (we will need to prove it)

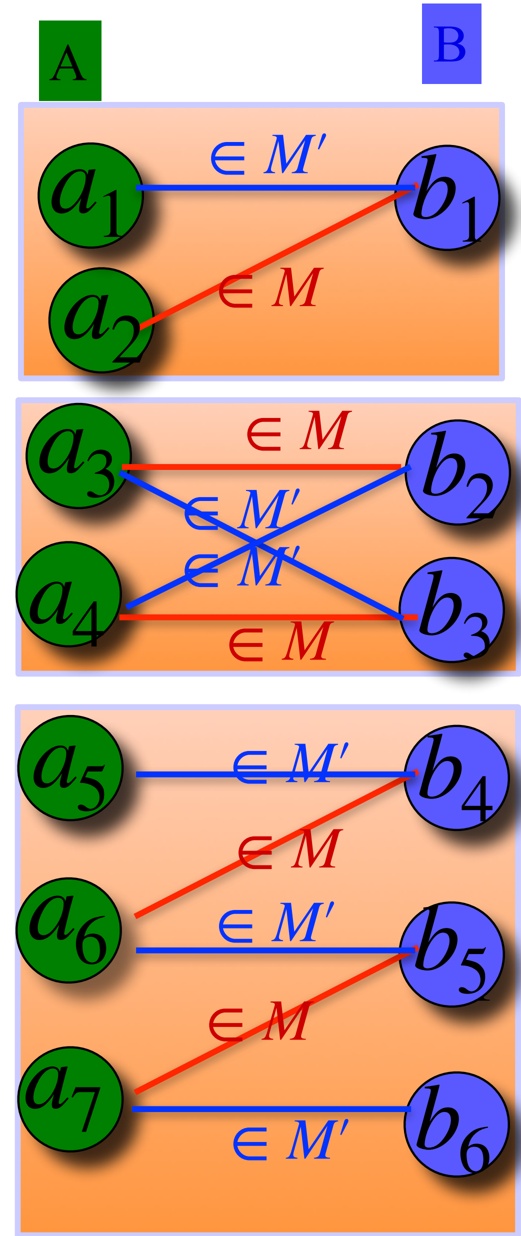
Running time: Each iteration, we increase |M| by 1, so the number of iterations is $\leq \min\{|A|,|B|\} \leq n$.

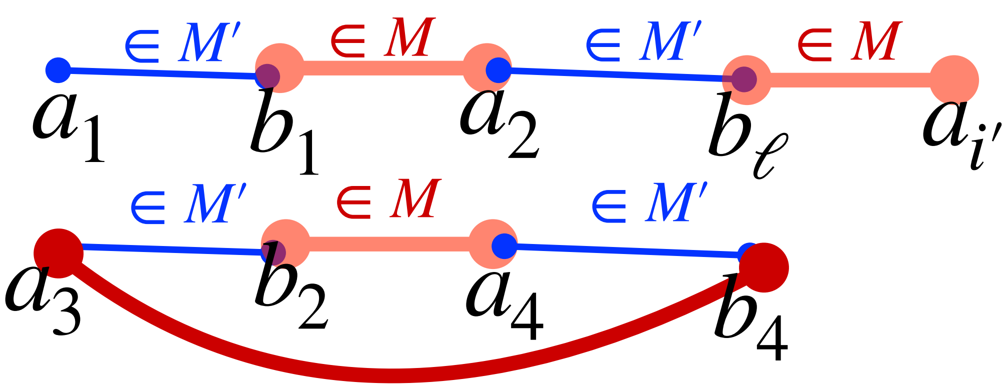
- Finding an augmented path is done via DFS or BFS, so its time is $O(|E|+|V|)$
- Overall time $O(|E||V|) = O(mn)$

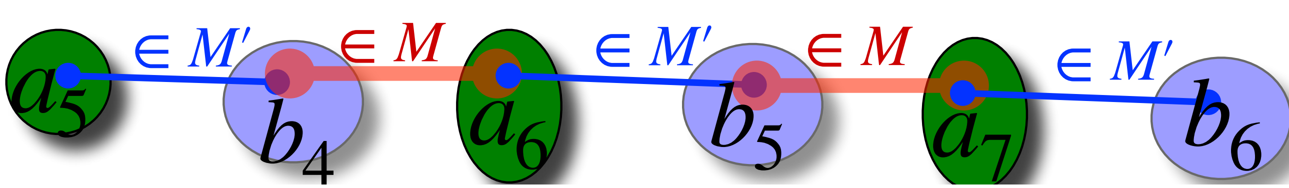
Optimality Theorem : M is maximum iff there is no augmenting path

Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \stackrel{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED







Notation

Definition. The ***value*** of a flow f, denoted by $|f|$, is given by

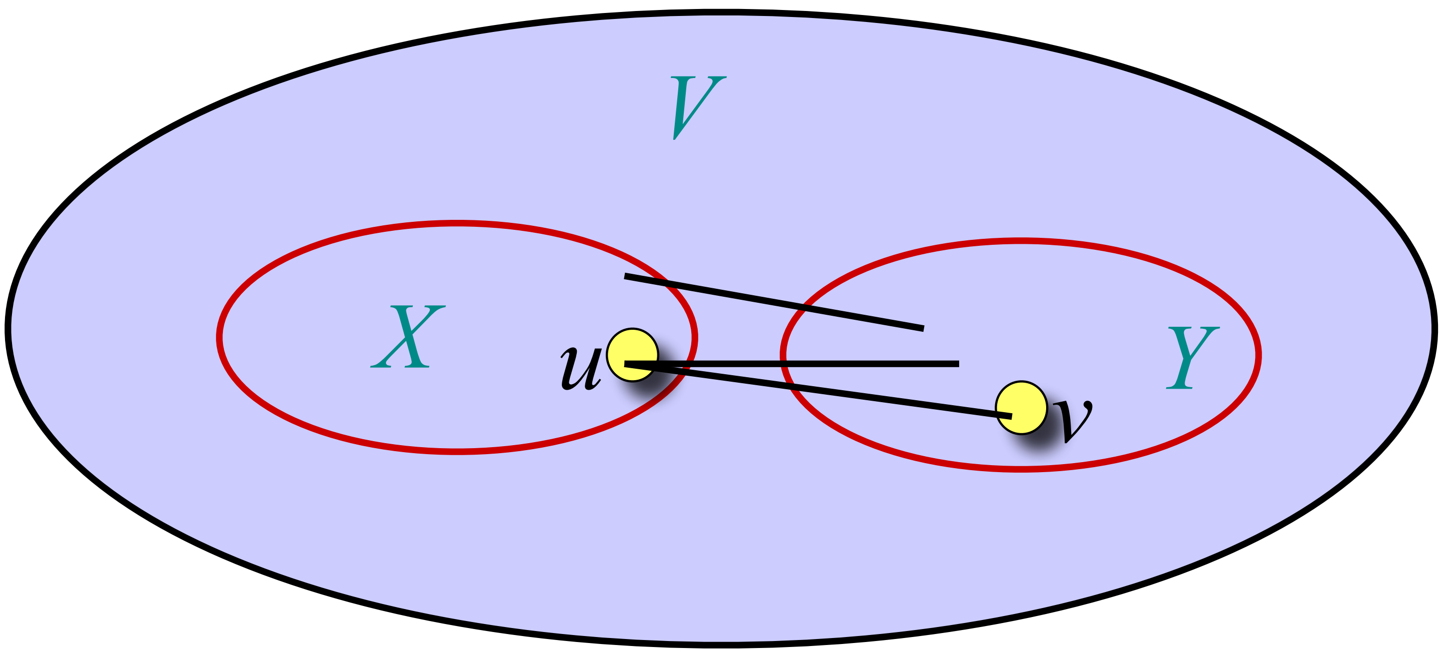
$$|f| = \sum_{v \in V} f(s, v) = f(s, V).$$

Implicit summation notation: A set used in an arithmetic formula represents a sum over the elements of the set.

- **Example** — flow conservation: $f(u, V) = \sum_{v \in V} f(u, v) = 0$ for all $u \in V - \{s, t\}$.

More definitions

Define $f(X, Y) = \sum_{u \in X} \sum_{v \in Y} f(u,v)$



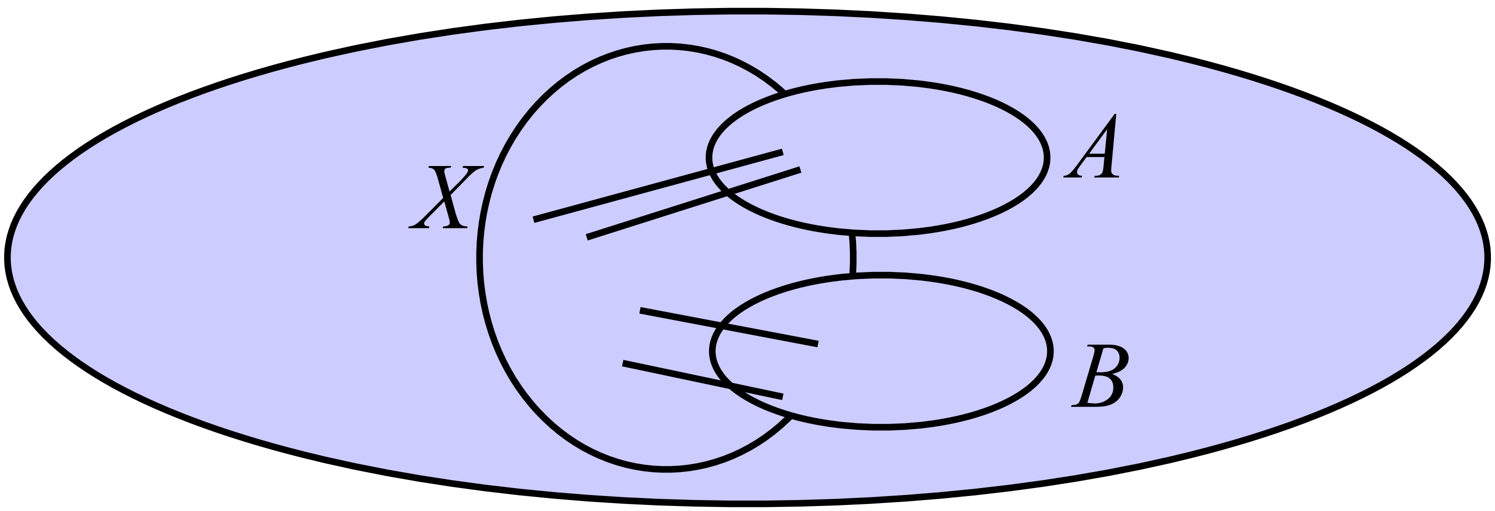
More properties of flow

Lemma:

1. If X does not contain s nor t, then $f(X, V) = 0$

   Proof: $f(X, V) = \sum_{u \in X} f(u, V) = \sum_{u \in X} 0.$

2. If A, B are **disjoint** sets of vertices, and X is another set, then $f(A \cup B, X) = f(A, X) + f(B, X)$



Note (property *): $f(A, X) = f(A \cup B, X) - f(B, X)$

And more properties of flow…

Lemma (Property #)**:**
For every set X of vertices

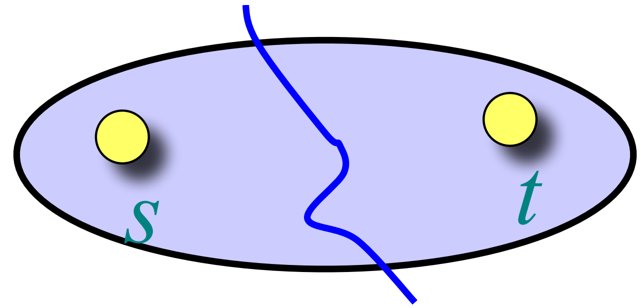
$$f(X,X)=0$$

Proof: $f(X,X) = \sum_{u \in X} \sum_{v \in X} f(u,v)$

and if $f(u, v)$ appears in the summation, then $f(v, u)$ also appears in the summation, and (skew-symmetric)

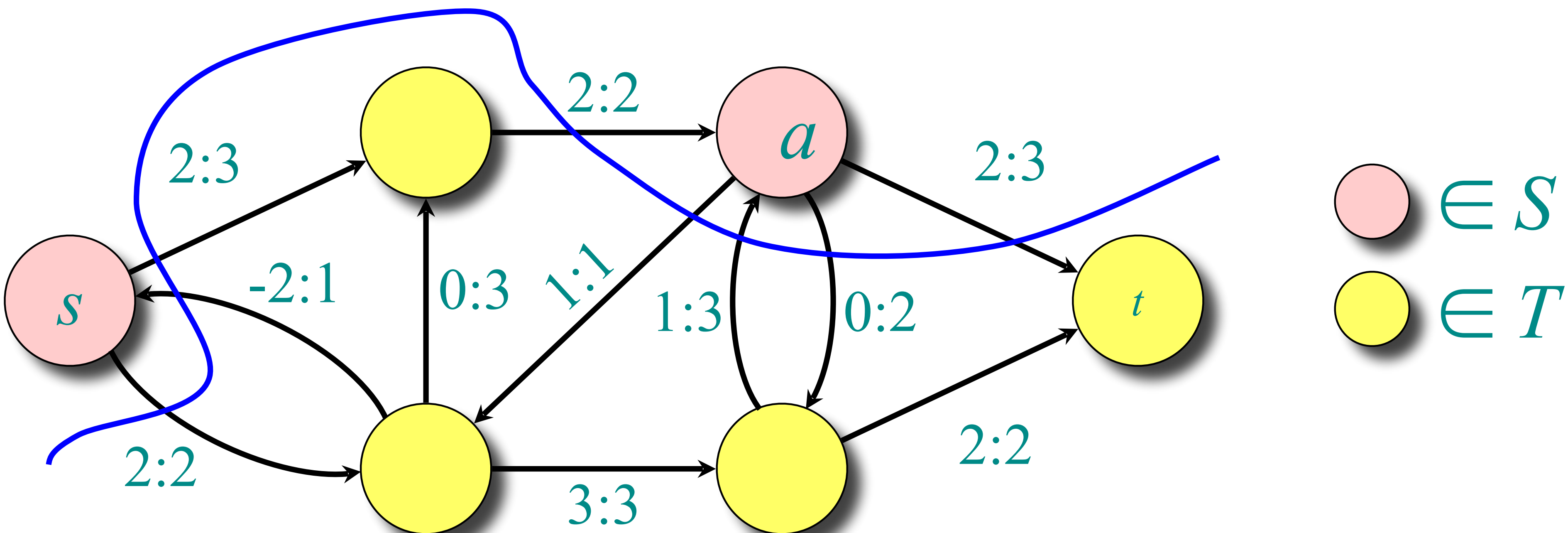
$$f(v,u) = -f(u,v).$$

Cuts



Definitions. A ***cut*** (S, T) of a flow network $G = (V, E)$ is a partition of V into two subsets S, T, such that $s \in S$ and $t \in T$.

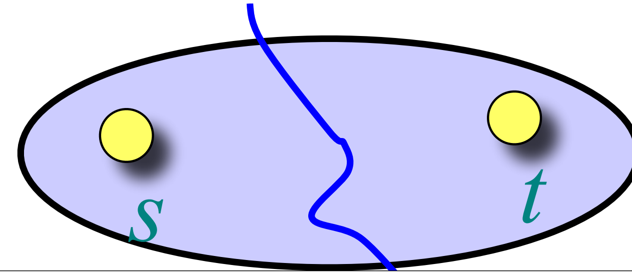
If f is a flow on G, then the ***flow across the cut*** is $f(S, T)$.



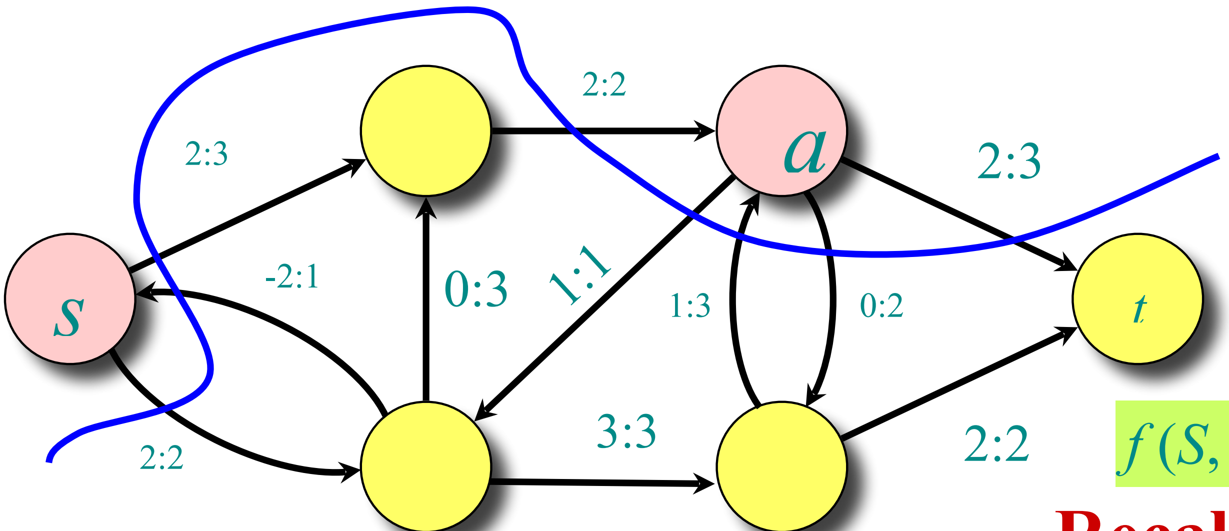
$S = \{s, a\}$

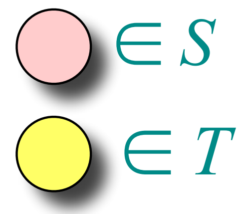
$f(S, T) = (2 + 2) + (-2 + 1 - 1 + 2) = 4$

Cuts



Definitions. A ***cut*** (S, T) of a flow network $G = (V, E)$ is a partition of V into two subsets S, T, such that $s \in S$ and $t \in T$.
If f is a flow on G, then the ***flow across the cut*** is $f(S, T)$.





$f(S, T) = (2 + 2) + (-2 + 1 - 1 + 2) = 4$

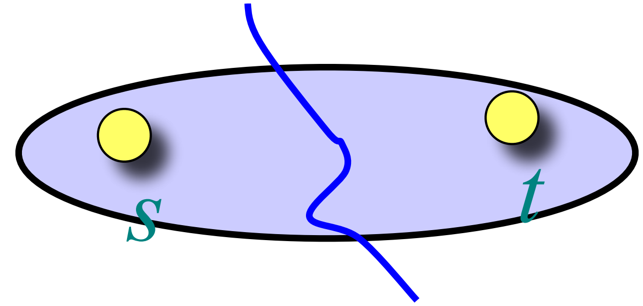
Recall: $|f| = f(s, V) = \sum_{v \in V} f(s, v)$

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

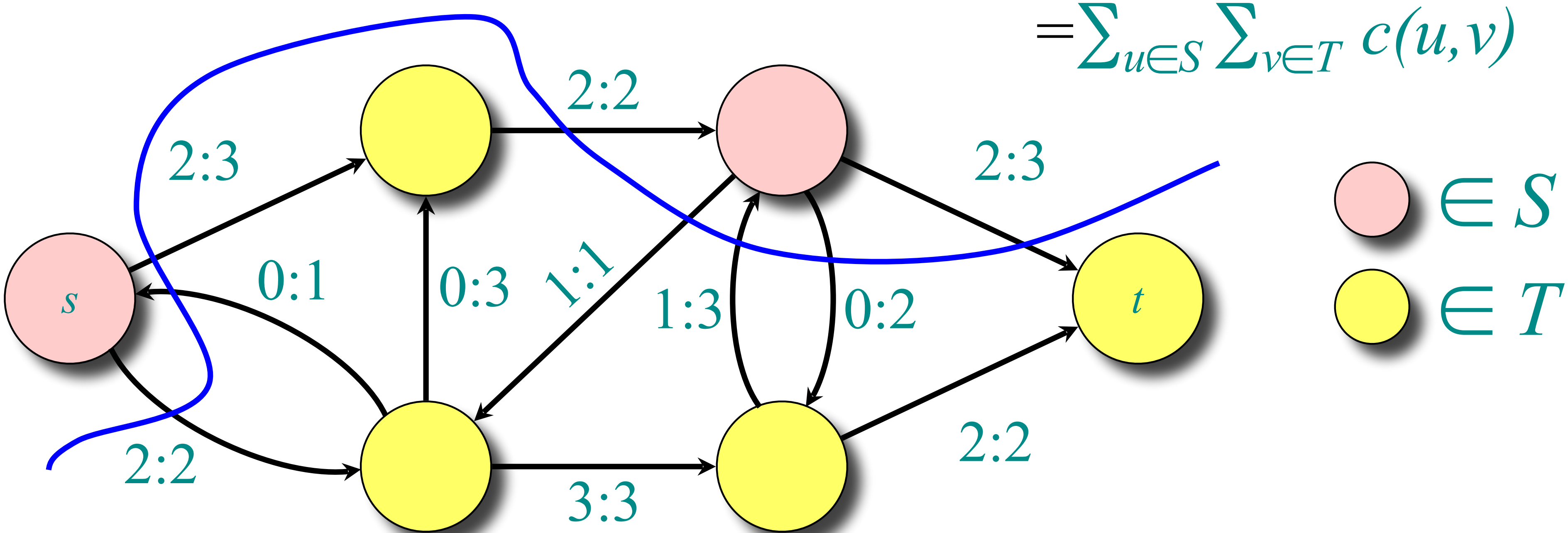
Proof.

$$
\begin{aligned}
f(S, T) &= f(S, V) - f(S, S) \quad \text{(property *)} \\
&= f(S, V) \\
&= f(s, V) + f(S - s, V) \\
&= f(s, V) = |f|.
\end{aligned}
$$

Capacity of a cut



Definition. The ***capacity of a cut*** (S, T) is $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$



$$c(S, T) = (3 + 2) + (1 + 2 + 3)$$
$$= 11$$

Upper bound on the maximum flow value

Theorem. The value of any flow no larger than the capacity of any cut: $|f| \le c(S,T)$.

Proof.

$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u \in S} \sum_{v \in T} f(u,v) \\
&\le \sum_{u \in S} \sum_{v \in T} c(u,v) \\
&= c(S,T) \quad \square
\end{aligned}
$$

The Max-flow, min-cut theorem

Theorem. The following conditions are are equivalent. If one is true the others are also true:

1. $|f| = c(S, T)$ for some cut (S, T). ← min-cut
2. f is a maximum flow.
3. f admits no augmenting paths.

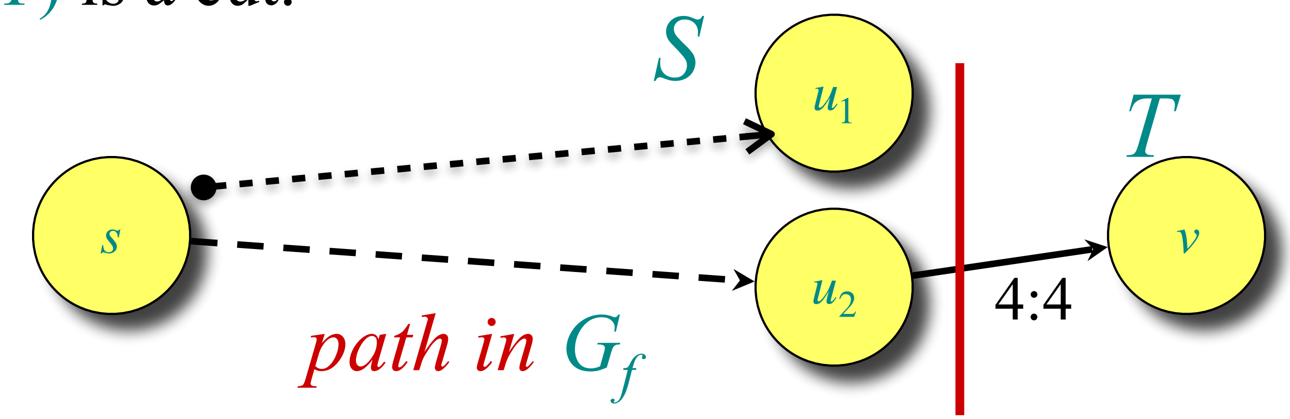
Proof.
(1) ⇒ (*2*): Since $|f| \leq c(S, T)$ for any cut (S, T) (by the theorem from a few slides back), the assumption that $|f| = c(S, T)$ implies that f is a maximum flow.

(*2*) ⇒ (*3*): If there were an augmenting path, the flow value could be increased, contradicting the maximality of f.

(3) ⇒ (1): Define $S = \{v \in V \mid \text{there exists a path in } G_f \text{ from } s \text{ to } v\}$,

- Let $T = V - S$. Since f admits no augmenting paths, there is no path from s to t in G_f.
- Hence, $s \in S$ and $t \notin S$, So $t \in T$.

- Thus (S, T) is a cut.



- Consider $u \in S$, $v \in T$. We must have $c_f(u, v) = 0$, since if $c_f(u, v) > 0$, then $v \in S$, not $v \in T$ as assumed.

- Thus, $f(u, v) = c(u, v)$, since $c_f(u, v) = c(u, v) - f(u, v)$.

- Summing over all $u \in S$ and $v \in T$ yields $f(S, T) = c(S, T)$, and since $|f| = f(S, T)$, the theorem follows.

Ford-Fulkerson max-flow algorithm

Algorithm:

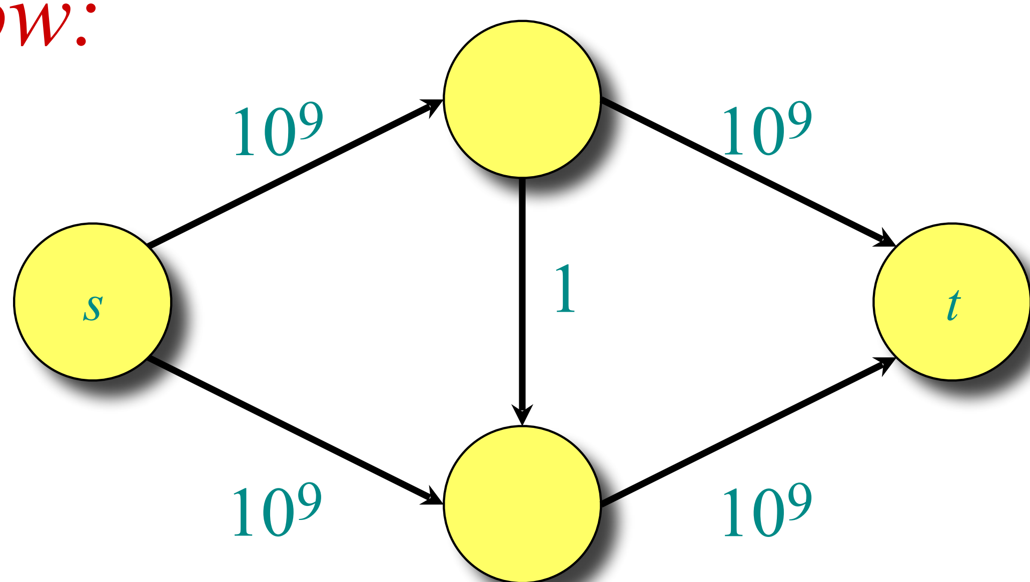
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G_f wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

Algorithm:

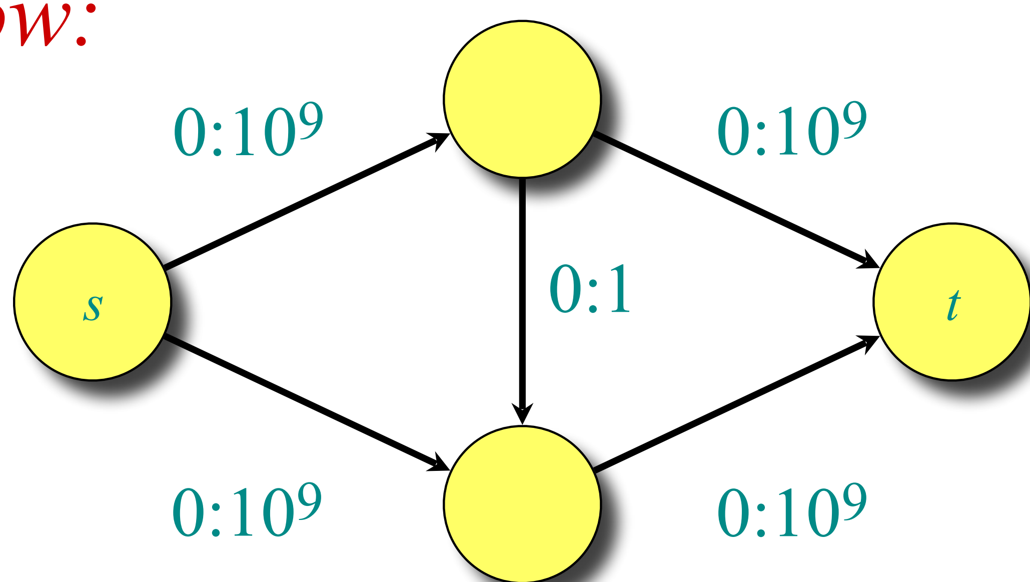
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

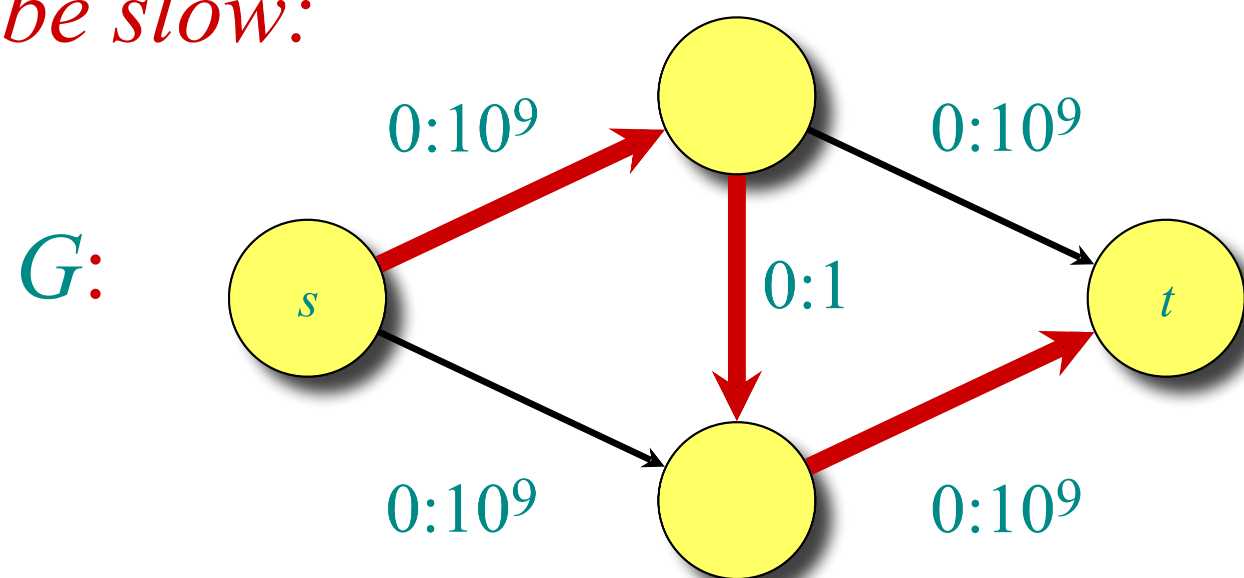
Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:



Ford-Fulkerson max-flow algorithm

Algorithm:

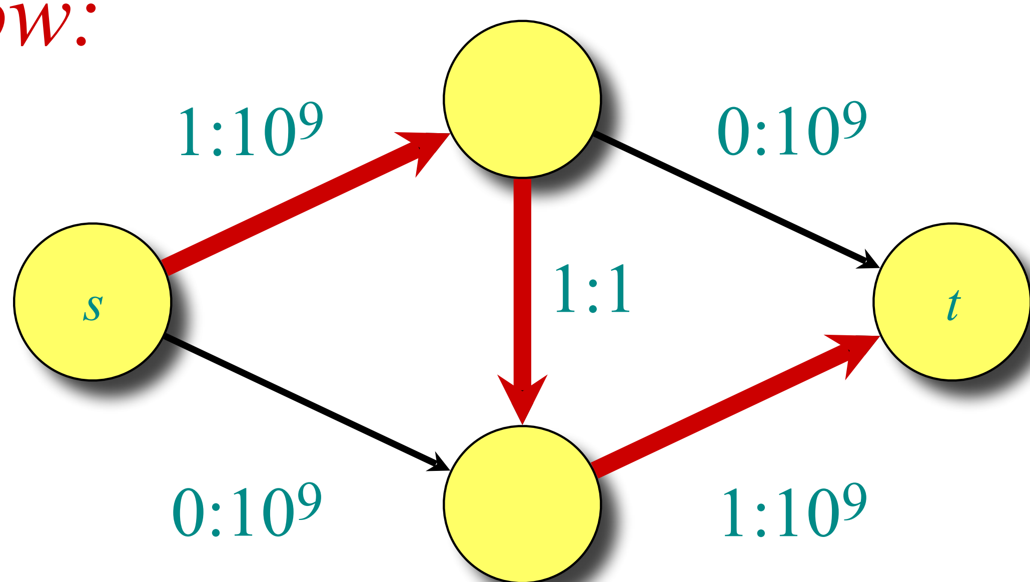
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

Algorithm:

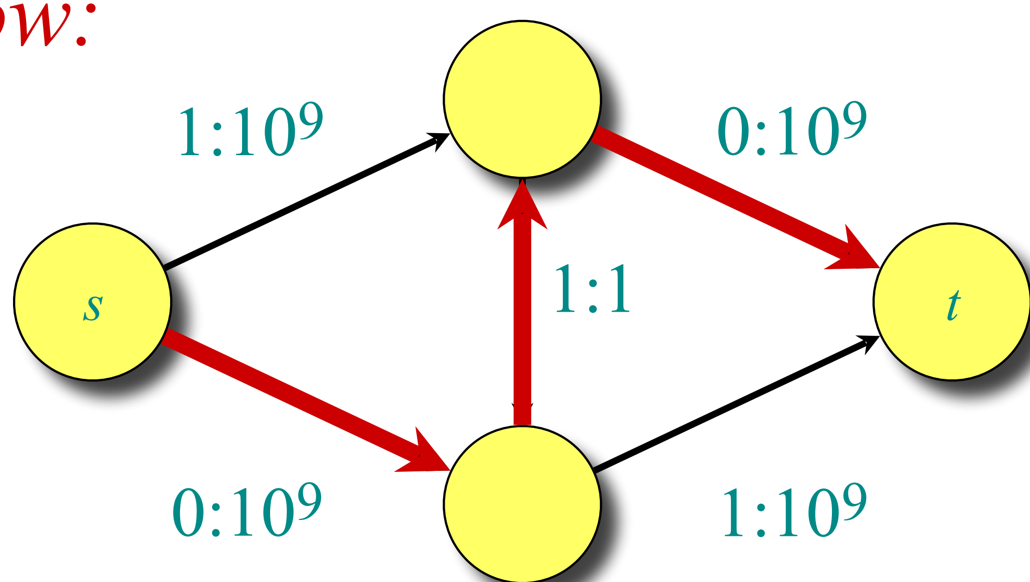
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

Algorithm:

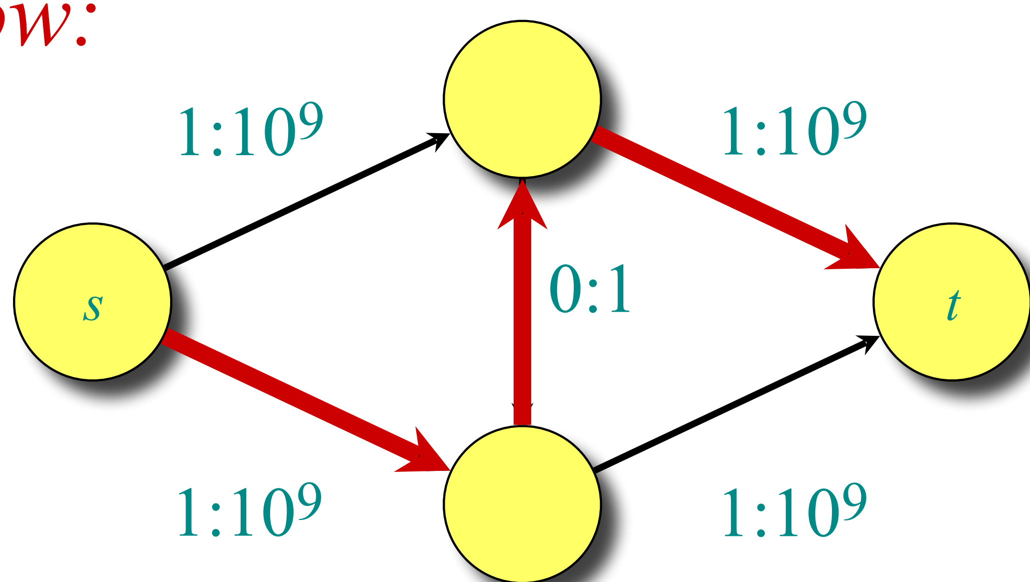
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

Algorithm:

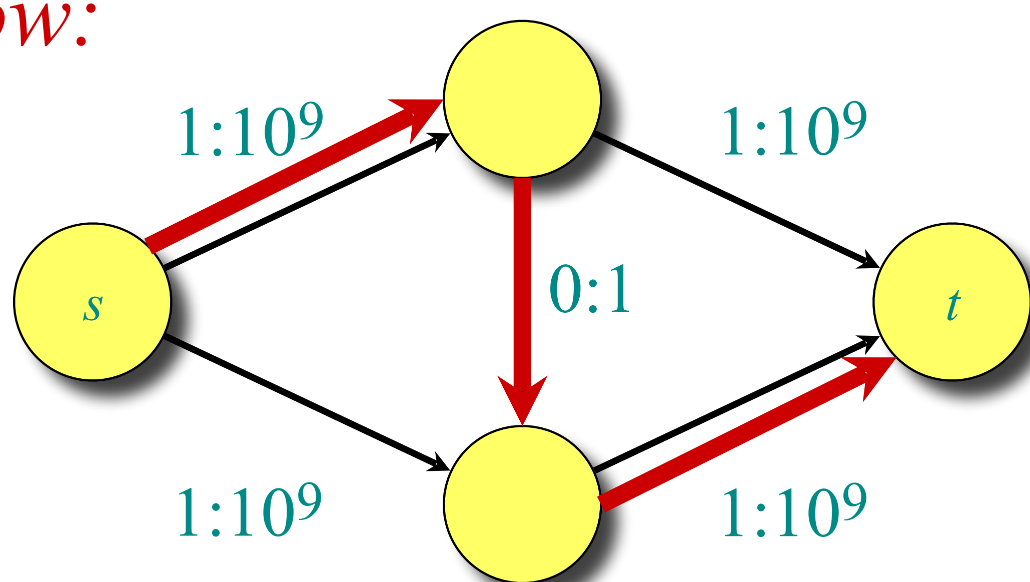
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:



Ford-Fulkerson max-flow algorithm

Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Runtime:

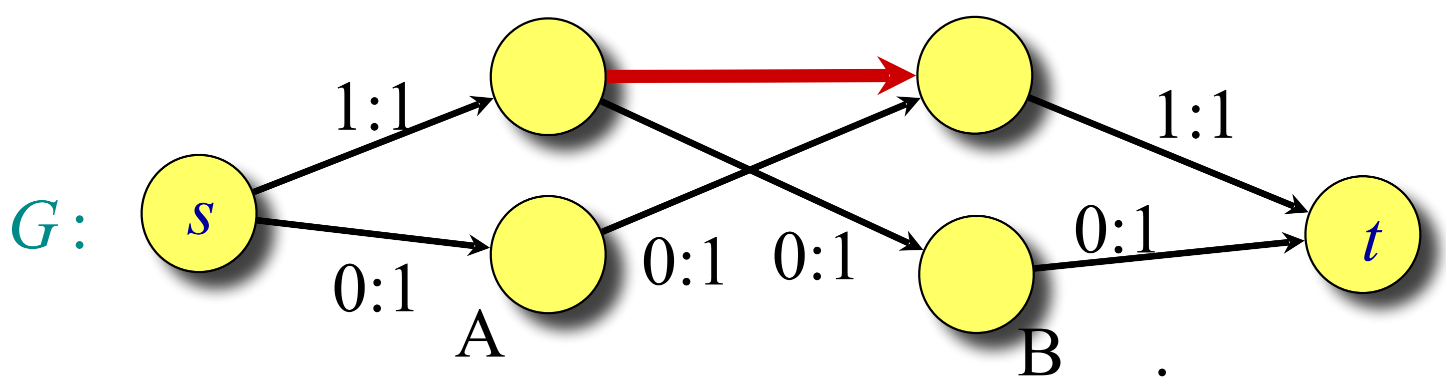
- Let $|f^*|$ be the value of a maximum flow, and assume it is an **integral** value.
- The initialization takes $O(|E|)$
- There are at most $|f^*|$ iterations of the loop
- Find an augmenting path with DFS in $O(|V|+|E|)$ time
- Each augmentation takes $O(|V|)$ time

$\Rightarrow O(|E| \cdot |f^*|)$ in total

Ford-Fulkerson and matching

Recall – we expressed the maximum matching problem as a network flow, but we can express the max flow as a matching, only if the flow is an **integer** flow.

However, this is always the case once using F&F algorithm: The flow along each edge is either 0 or 1.



Runtime analysis of F&F-algorithm applied for matching

- We saw that in each iteration of F&F algorithm, $|f|$ increases by at least 1.
- Let $|f^*|$ be the maximum value.
- How large can $|f^*|$ be ?

- Claim: $|f^*| \leq min\{|A|, |B|\}$ (why ?)
- Runtime is $O(|E| \cdot min\{|A|, |B|\}) = O(|E||V|)$
- Can be done in $O(|E|^{1/2} \cdot |V|)$ (Dinic Algorithm)

Edmonds-Karp algorithm

Edmonds and Karp noticed that many people's implementations of Ford-Fulkerson augment along a ***breadth-first augmenting path***: a path with smallest number of edges in G_f from s to t.

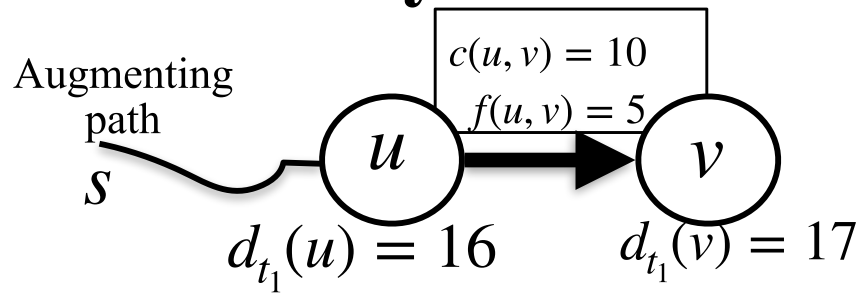
These implementations would always run relatively fast.

Since a breadth-first augmenting path can be found in $O(|E|)$ time, their analysis, focuses on bounding the number of flow augmentations.

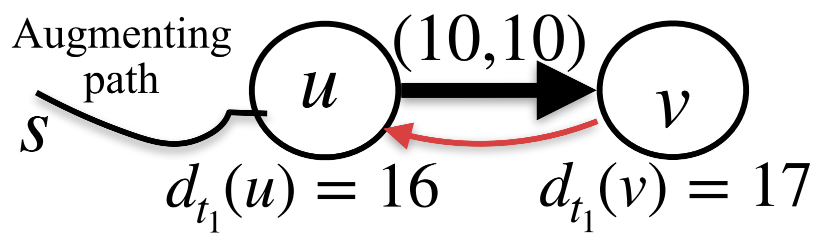
(In independent work, Dinic also gave polynomial-time bounds.)

Edmonds-Karp algorithm - analysis

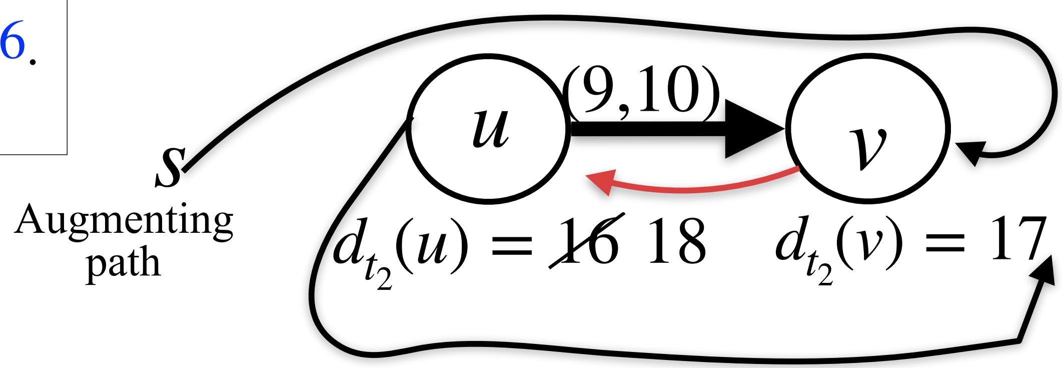
- EK consists of iterations. An iterations consists of building G_f, finding a min-hoops augmenting path, and augment. Let f_j be the flow after the j'th iteration. ($j = 0,1,2\ldots$). We want to bound the max number of iterations.
- An edge $(u, v) \in E$ is **saturated** iff either $f(u, v) = 0$ or $f(u, v) = c(u, v)$. At each iteration of EK, at least one edge becomes saturated.
- Let $d_j(v)$ be shortest path from s to vertex v, in G_{f_j} (that is, after j iteration of EK)
- Lemma1: For every $v \in V$, $d_j(v) \leq d_{j+1}(v)$
- The argument uses induction, plus the mechanism by which an edge starts/stop being saturated.
- Assume that an edge $(u, v) \in E$ is on an augmented path in at t_1. Assume $d_{t_1}(\mathbf{u}) = 16$. This means that $d_{t_1}(\mathbf{v}) = 16 + 1 = 17$.



Assume after augmentation, (u, v) is saturated

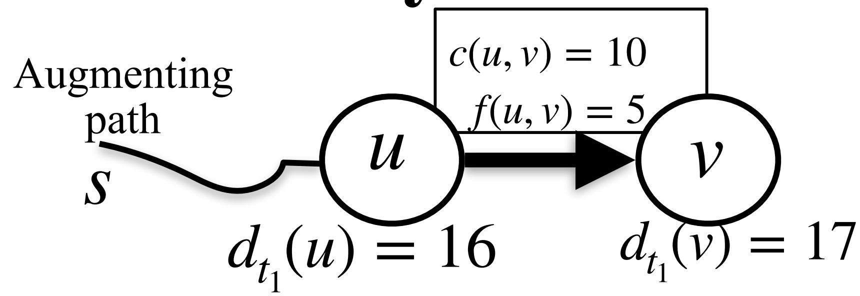


If in a later iteration $t_2 > t_1$ the edge (u, v) will be used again (in this direction), some of the flow must be canceled via flow in opposite direction, again via augmenting path. Since $d_{t_2}(v) = 17$ and now the path is from $v \rightarrow u$, it must be that $d_{t_2}(u) \geq 1 + d_{t_2}(v) \geq 17$

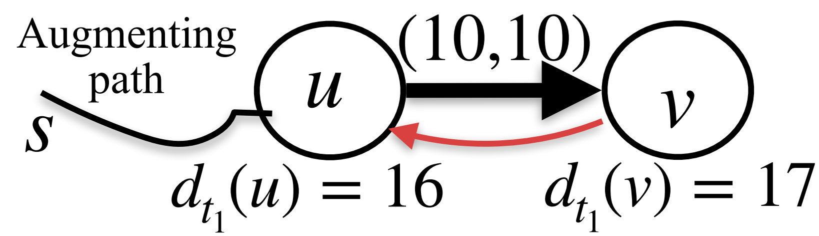


Edmonds-Karp algorithm - analysis

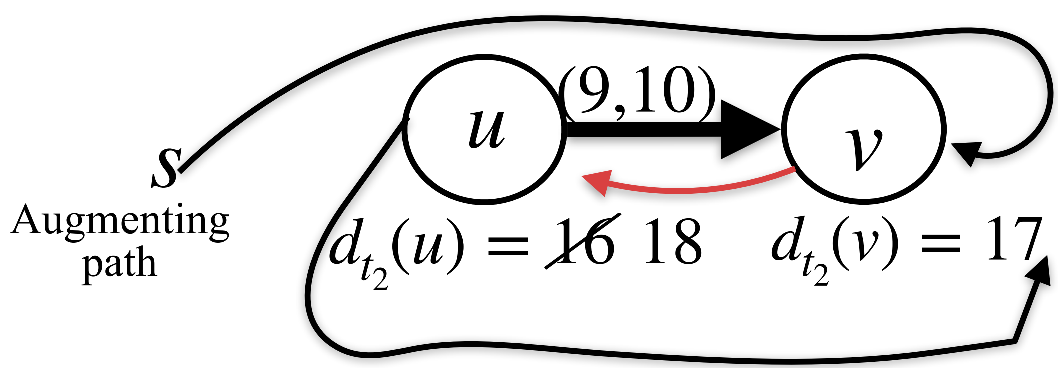
- Note that $\max d_i(v) = n - 1$. This is the max distance of any path in a graph with n nodes
- Each EK iterations forces one edge (u, v) to become saturated. To reuse it, we must increase $d_j(v)$ by 2.
- Each iteration if EK that uses this edge, at least one vertex has increase its distance by 2
- So this edge could become saturated $\leq n/2$ times
- Therefor, the number of iterations is $\leq mn$
- Each iteration takes $O(m)$
- Total time $O(m^2 n)$
- Note that $m \leq \binom{n}{2}$, so running time could be $O(n^5)$:-(



Assume after augmentation, (u, v) is saturated



If in a later iteration $t_2 > t_1$ the edge (u, v) will be used again (in this direction), some of the flow must be canceled via flow in opposite direction, again via augmenting path. Since $d_{t_2}(v) = 17$ and now the path is from $v \rightarrow u$, it must be that $d_{t_2}(u) \geq 1 + d_{t_2}(v) \geq 17$



Running time of Edmonds-Karp

- One can show that if BFS is used, then the number of flow augmentations (i.e., the number of iterations of the while loop) is $O(|V|\ |E|)$.
- Breadth-first search runs in $O(|E|)$ time
- All other bookkeeping is $O(|V|)$ per augmentation.

$\Rightarrow$ The Edmonds-Karp maximum-flow algorithm runs in $O(|V| \cdot |E|^2)$ time.

=> Dinitz (Dinic) Algorithm runs in

Lets prove