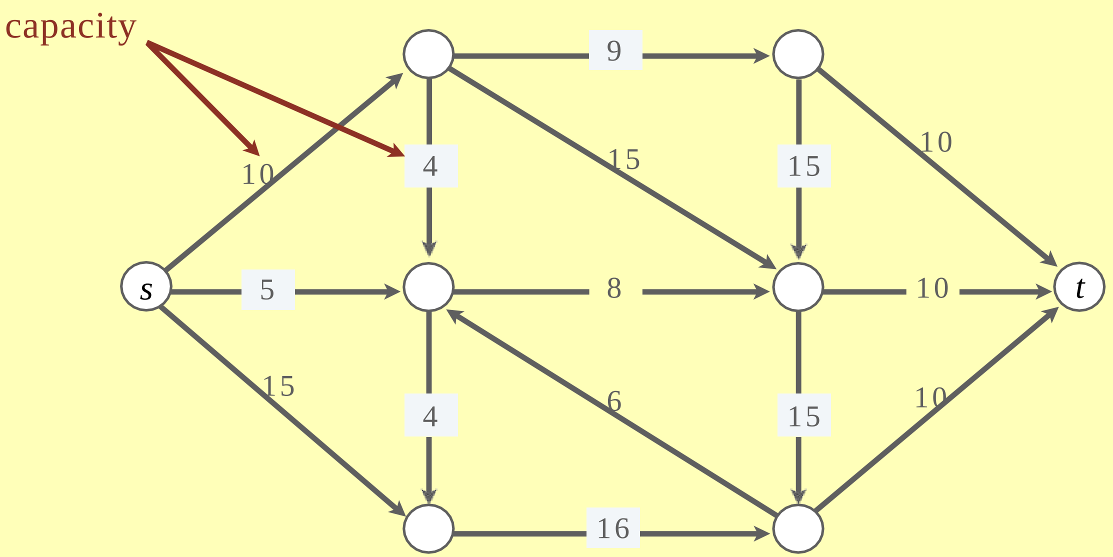
Network Flows and Bipartite Matching

Rangaballav Pradhan
ITER, SOADU

Flow Network

- A Flow Network is a special type of transportation network where some materials (traffic) flows across the nodes and through the edges.
- For example:
  - a road transportation system in which the edges are road segments and the nodes are junctions;
  - a computer network in which the edges are links that can carry data packets and the nodes are switches;
  - a fluid network in which edges are pipes that carry liquid, and the nodes are junctures where pipes are plugged together.

Intuition. Material flowing through a transportation network; material originates at source (s) and is sent to sink (t).

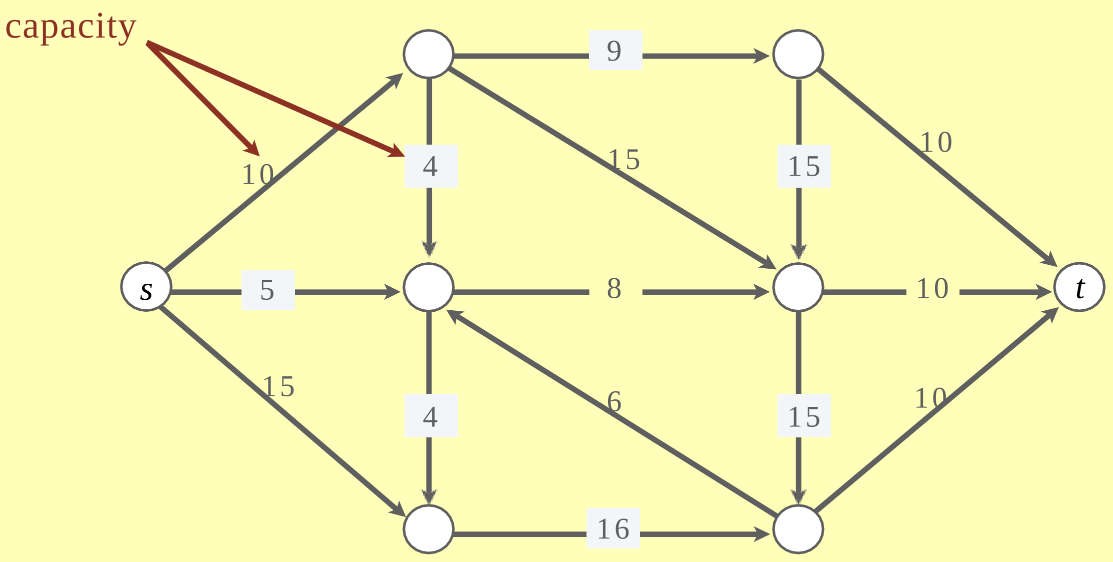


Flow Network

- A Flow Network is a special type of transportation network where some materials (traffic) flows across the nodes and through the edges.
- A flow network is associated with several ingredients:
  - ➢ capacities on the edges, indicating how much they can carry;
  - ➢ source nodes in the graph, which generate traffic;
  - ➢ sink or destination nodes in the graph, which can "absorb" traffic as it arrives;
  - ➢ the traffic (flow) itself, which is transmitted across the edges.

Assumptions.

- ➢ A single source node (s) with no in-degree and single sink (t) node with no out-degree.
- ➢ At least one edge incident to each node;
- ➢ All capacities are integers.

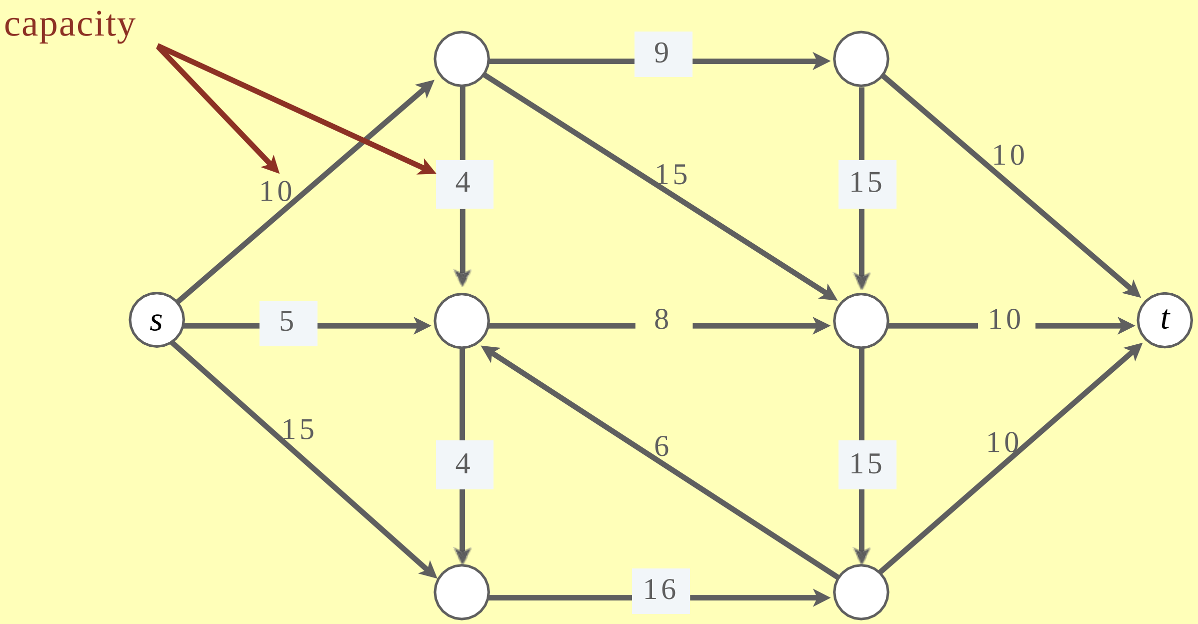


Flow Network: Definition

A Flow Network is a tuple $G = (V, E, s, t, c)$.

- Digraph (V, E) with source $s \in V$ and sink $t \in V$.
- Capacity $c(e) \geq 0$ for each $e \in E$.

assume all nodes are reachable from s



Flow: Definition

Def. An *st*-flow (flow) $f : E \rightarrow R$ is a function that satisfies:

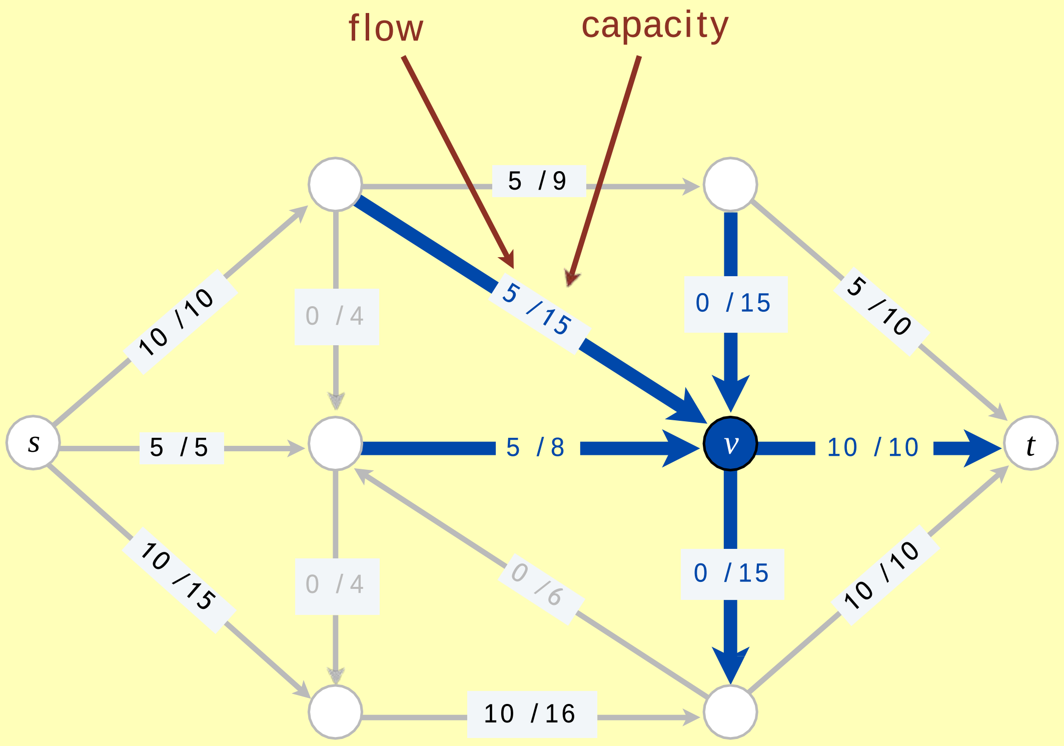
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity constraint]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation constraint]

Inflow (v) Outflow (v)

- the value $f(e)$ represents the amount of flow carried by edge e.
- Inflow (v): the total flow value $f(e)$ over all edges entering node v.
- Outflow (v): the total flow value $f(e)$ over all edges leaving node v.

inflow at v = 5 + 5 + 0 = 10

outflow at v = 10 + 0 = 10

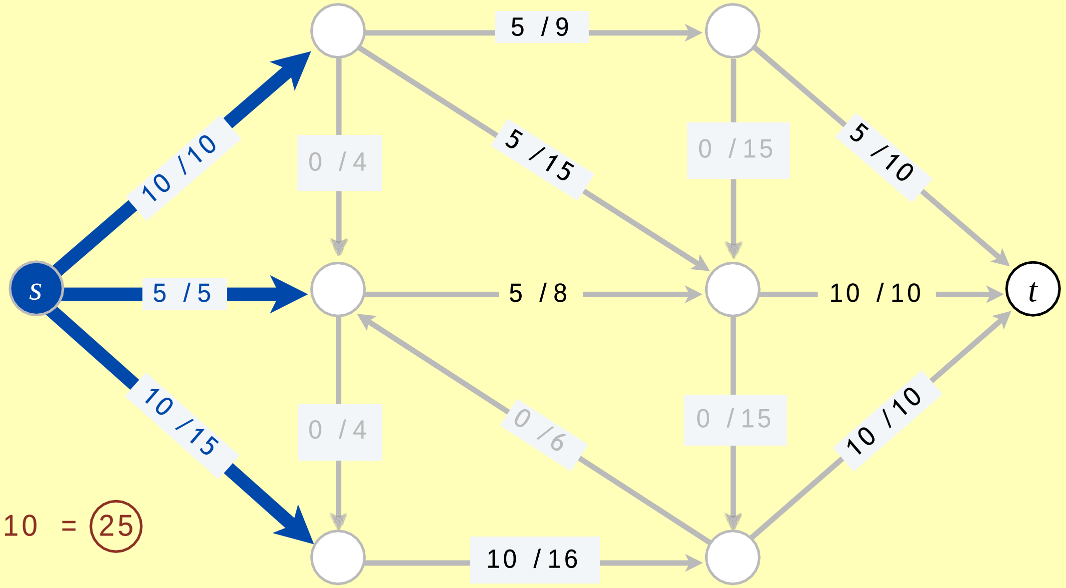


Value of Flow

Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$



value = 5 + 10 + 10 = 25

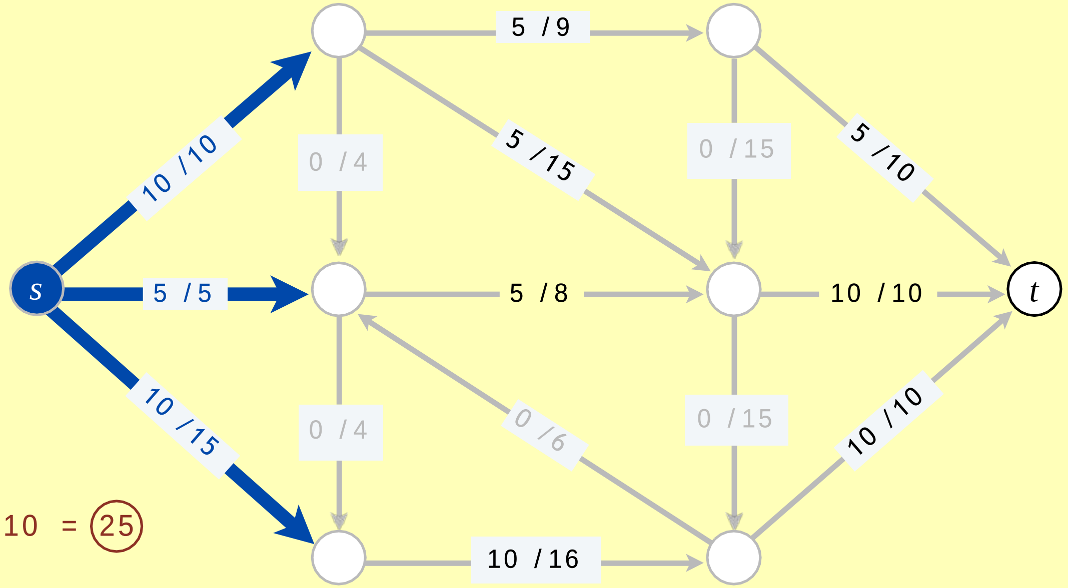
Maximum-Flow Problem

Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

Max-flow problem. Given a Flow network $G = (V, E, s, t, c)$, Find a flow of maximum value.



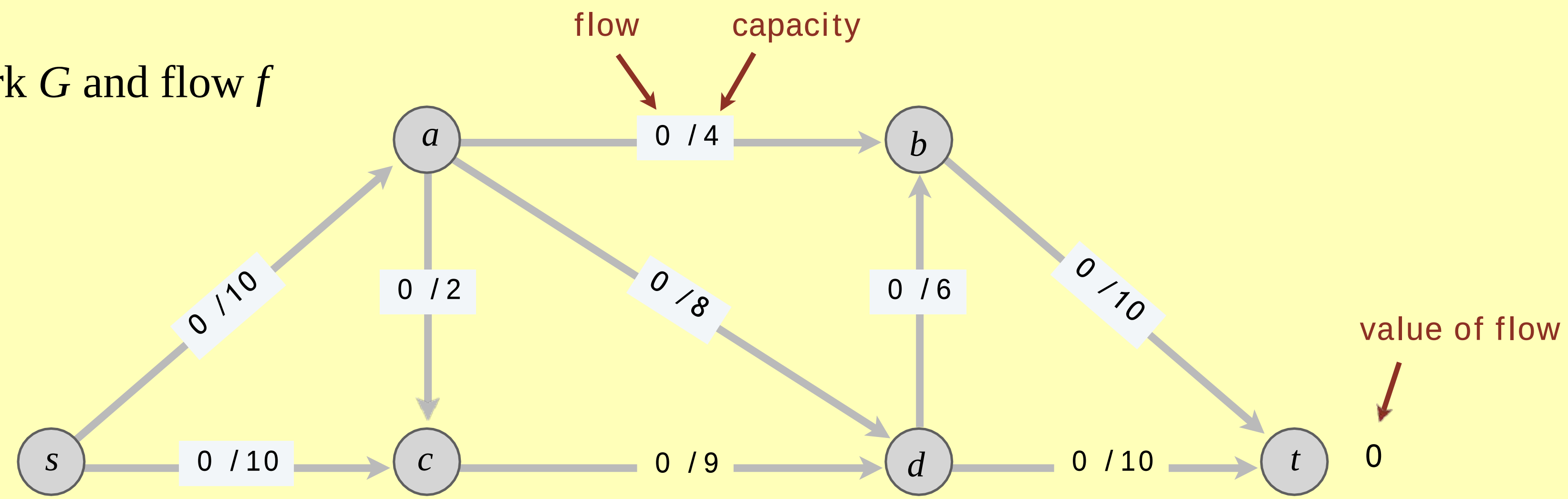
value = 5 + 10 + 10 = 25

Towards a max-flow algorithm

Greedy algorithm.

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

Flow Network G and flow f

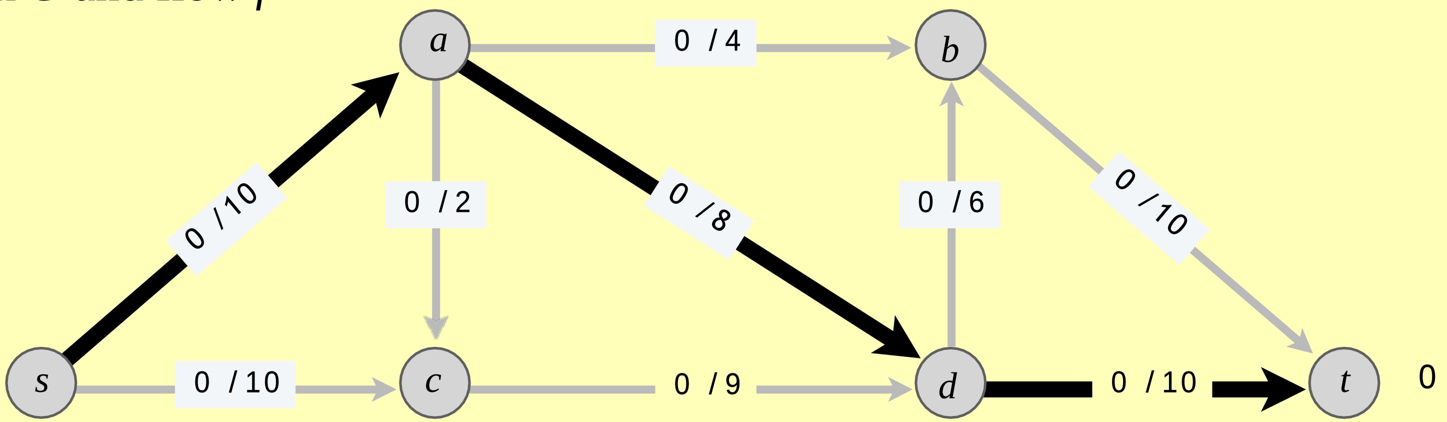


Towards a max-flow algorithm

Greedy algorithm.

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

Flow Network G and flow f

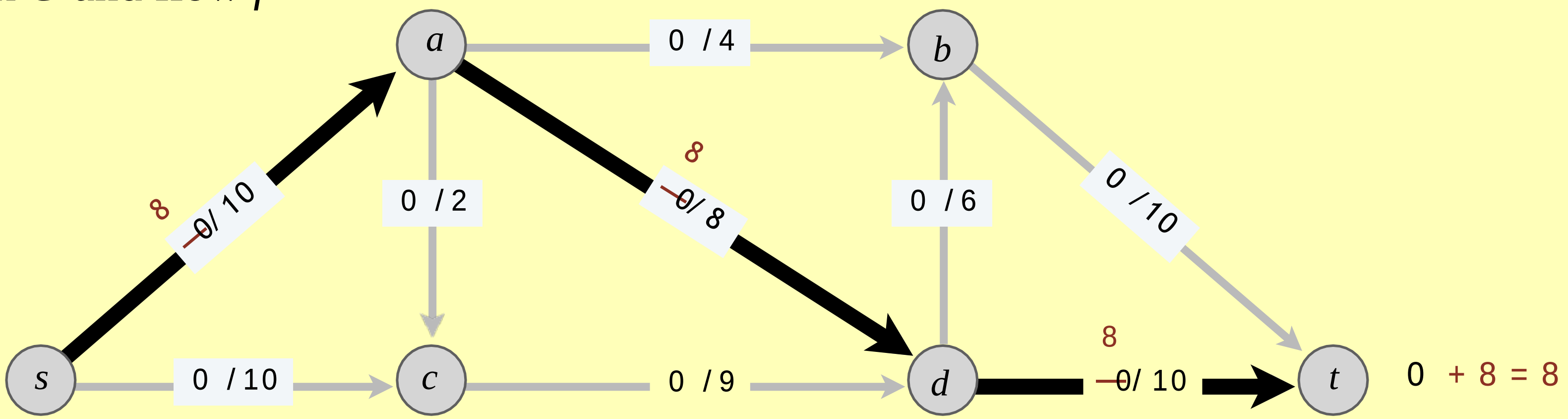


Towards a max-flow algorithm

Greedy algorithm.

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

Flow Network G and flow f

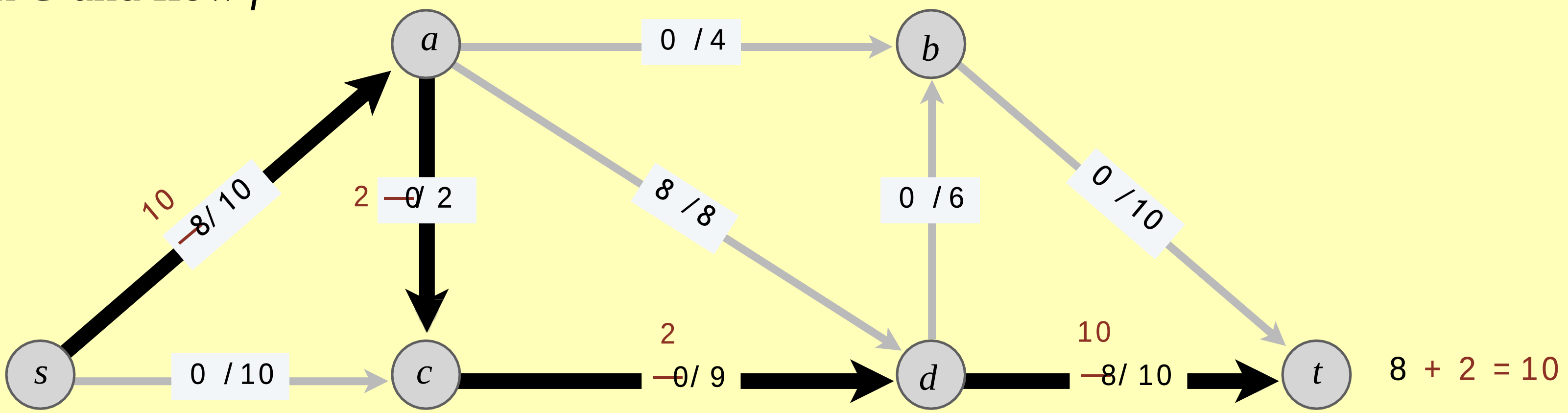


Towards a max-flow algorithm

Greedy algorithm.

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

Flow Network G and flow f

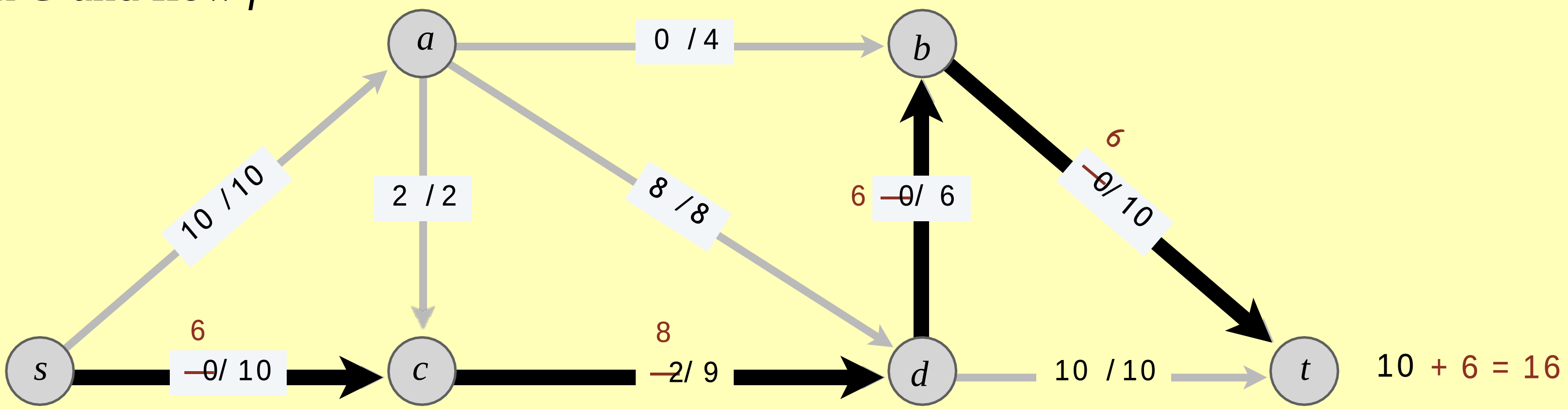


Towards a max-flow algorithm

Greedy algorithm.

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

Flow Network G and flow f



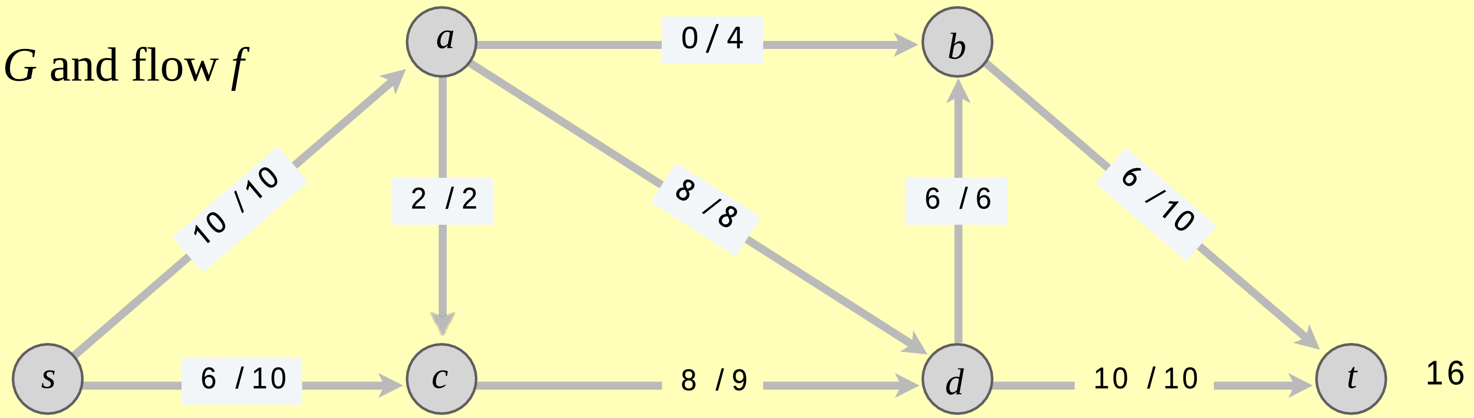
Towards a max-flow algorithm

Greedy algorithm.

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

Ending Flow value = 16

Flow Network G and flow f



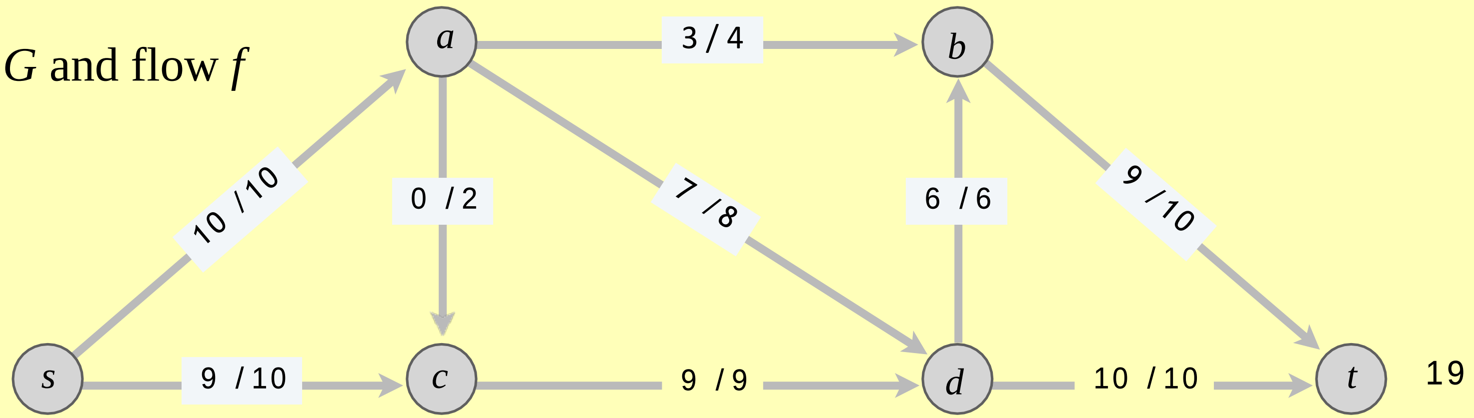
Towards a max-flow algorithm

Greedy algorithm.

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

But maximum flow value = 19

Flow Network G and flow f



Why the greedy algorithm fails?

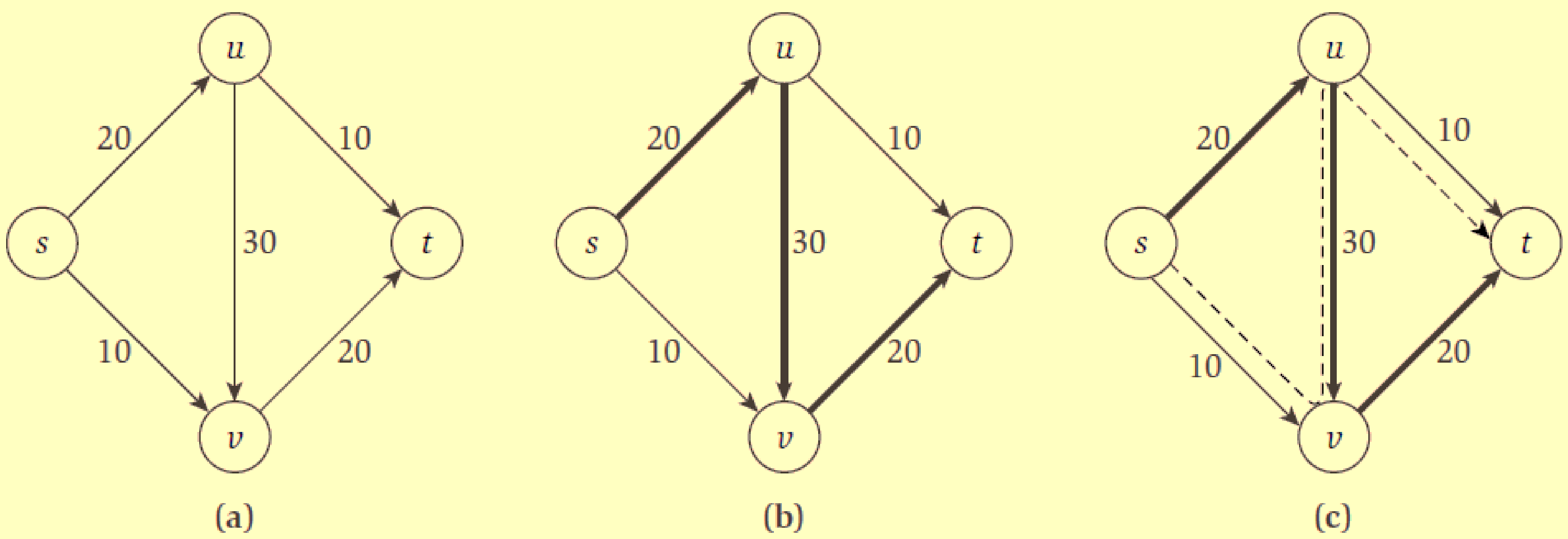
Q. Why does the greedy algorithm fail?

A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex. Consider flow network G.

- Greedy algorithm could choose $s \to \upsilon \to v \to t$ as first path.

Flow Network G



Bottom line. Need some mechanism to “undo” a bad decision.

Residual network

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

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

Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

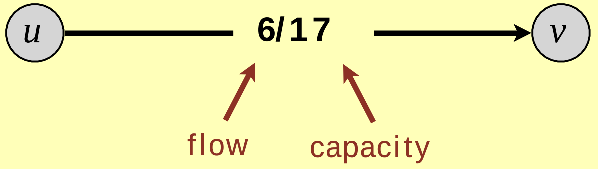
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^{\text{reverse}} \in E \end{cases}$$

Residual network. $G_f = (V, E_f, s, t, c_f)$.

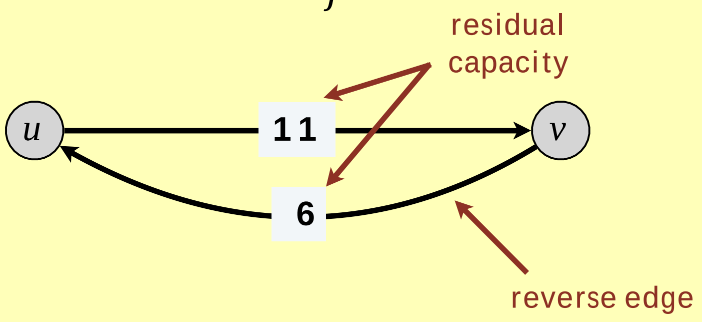
- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\text{reverse}} : f(e) > 0\}$.

edges with positive residual capacity

Original Flow Network G



Residual Flow Network G_f



Augmenting path

Def. An augmenting path is a simple $s \rightsquigarrow t$ path in the residual network G_f.

Def. The bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling $f' \leftarrow$ AUGMENT(f, c, P), the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

```
AUGMENT(f , c, P)
    Let b = bottleneck(P, f )
    For each edge (u, v) ∈ P
        If e = (u, v) is a forward edge then
            increase f (e) in G by b
        Else ((u, v) is a backward edge, and let e = (v, u))
            decrease f (e) in G by b
        Endif
    Endfor
    Return(f)
```

Ford–Fulkerson algorithm

Ford–Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find a simple $s \rightsquigarrow t$ path P in the residual network G_f.
- Augment flow along path P.
- Repeat until you get stuck.

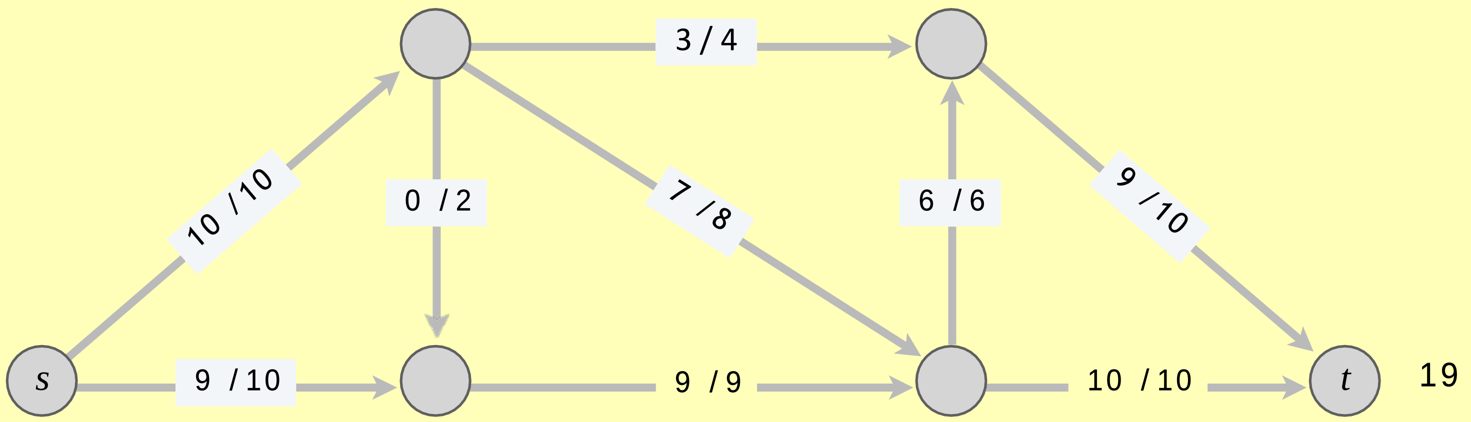
FORD–FULKERSON(G)

FOREACH edge $e \in E$:
  $f(e) \leftarrow 0$.
$G_f \leftarrow$ residual network of G with respect to flow f.
WHILE (there exists an $s \rightsquigarrow t$ path P in G_f)
  $f \leftarrow$ AUGMENT(f, c, P).
  Update G_f.
RETURN f.

augmenting path

Time = $O(E^*f)$

Example



Ford–Fulkerson algorithm: Analysis

- Let $m = |E|$ and $n = |V|$
- The *for* loop runs for O(m) times.
- The algorithm terminates in at most C iterations of the While loop in the worst case, where $C = \sum_{e \text{ out of } s} c(e)$.
- The residual graph G_f has at most $2m$ edges. Given the new flow f , we can build the new residual graph in $O(m + n)$ time

FORD–FULKERSON(G)

FOREACH edge $e \in E$:
    $f(e) \leftarrow 0$.
$G_f \leftarrow$ residual network of G with respect to flow f.
WHILE (there exists an s⤳t path P in G_f)
    $f \leftarrow$ AUGMENT(f, c, P).
    Update G_f.
RETURN f.

augmenting path

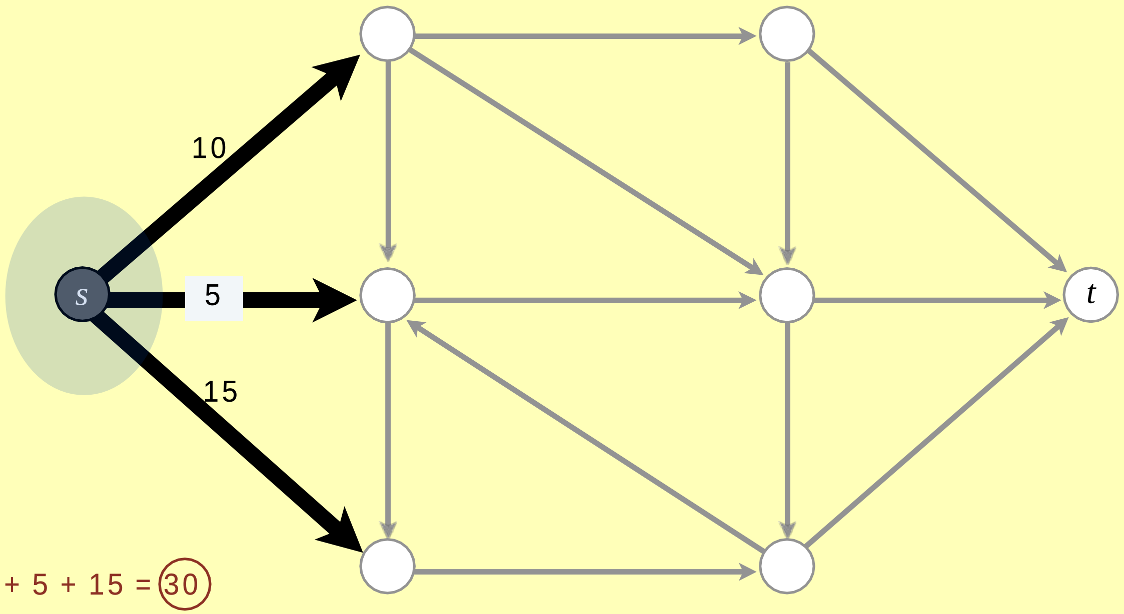
- We can maintain G_f using an adjacency list representation in $O(m + n)$ time.
- To find an s-t path in G_f , we can use BFS or DFS, which run in $O(m + n)$ time;
- By our earlier assumption that each vertex is incident with at least one edge, i.e., $m \geq n/2$. So, $O(m + n)$ is the same as $O(m)$.
- The procedure AUGMENT(f , c, P) takes time $O(n)$, as the path P has at most $n - 1$ edges.
- So, the total time complexity in worst case: $O(E*f)$ or $O(m*C)$

Cut in a Flow Network

Def. An *st*-cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

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

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



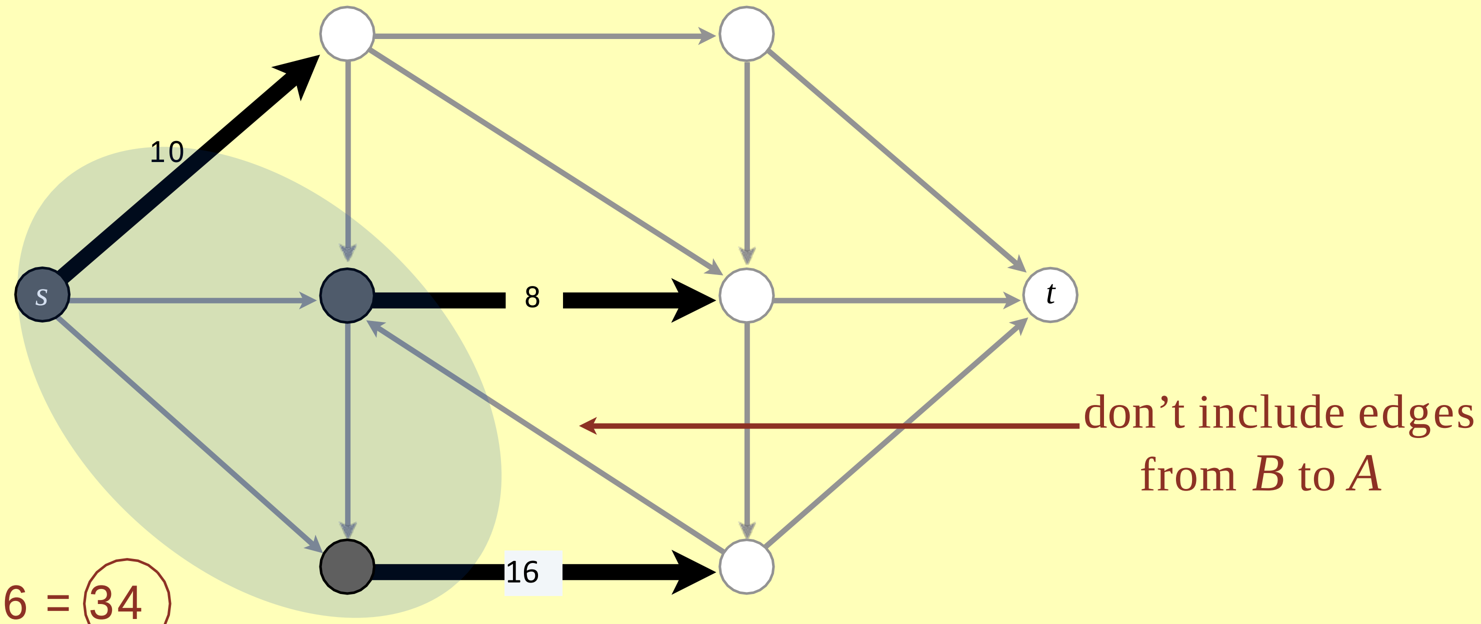
capacity = 10 + 5 + 15 = 30

Cut in a Flow Network

Def. An *st*-cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

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

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



capacity = 10 + 8 + 16 = 34

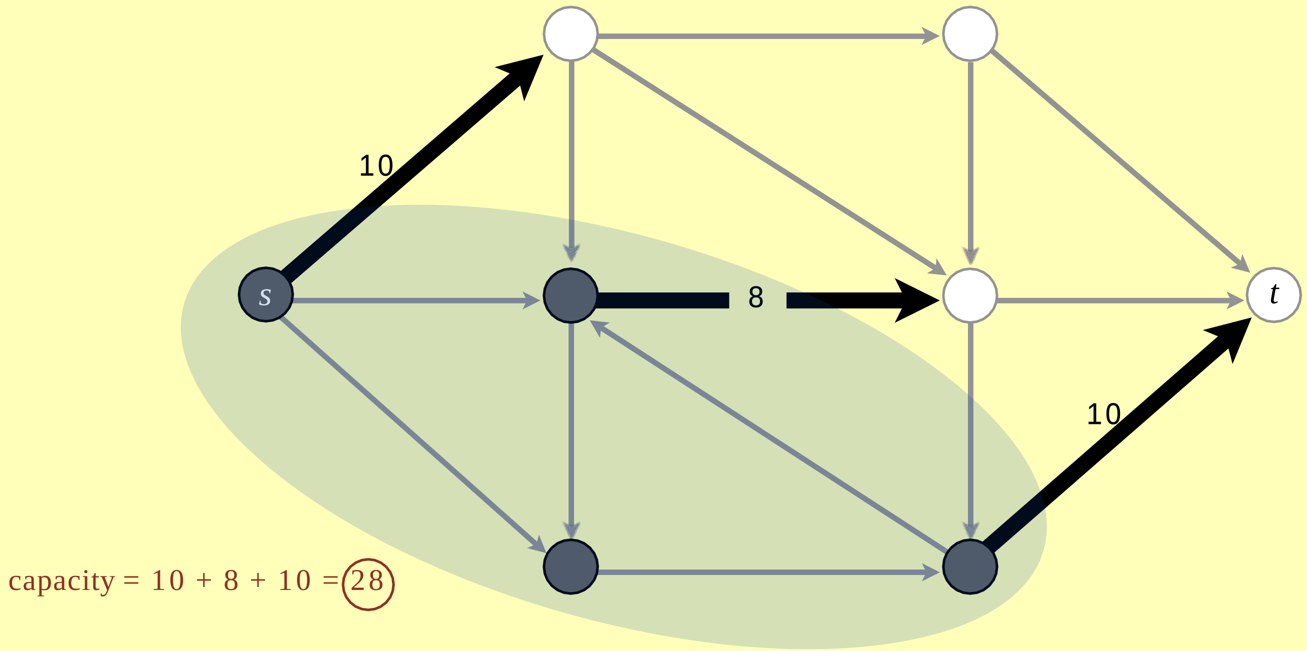
Minimum-cut problem

Def. An *st*-cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

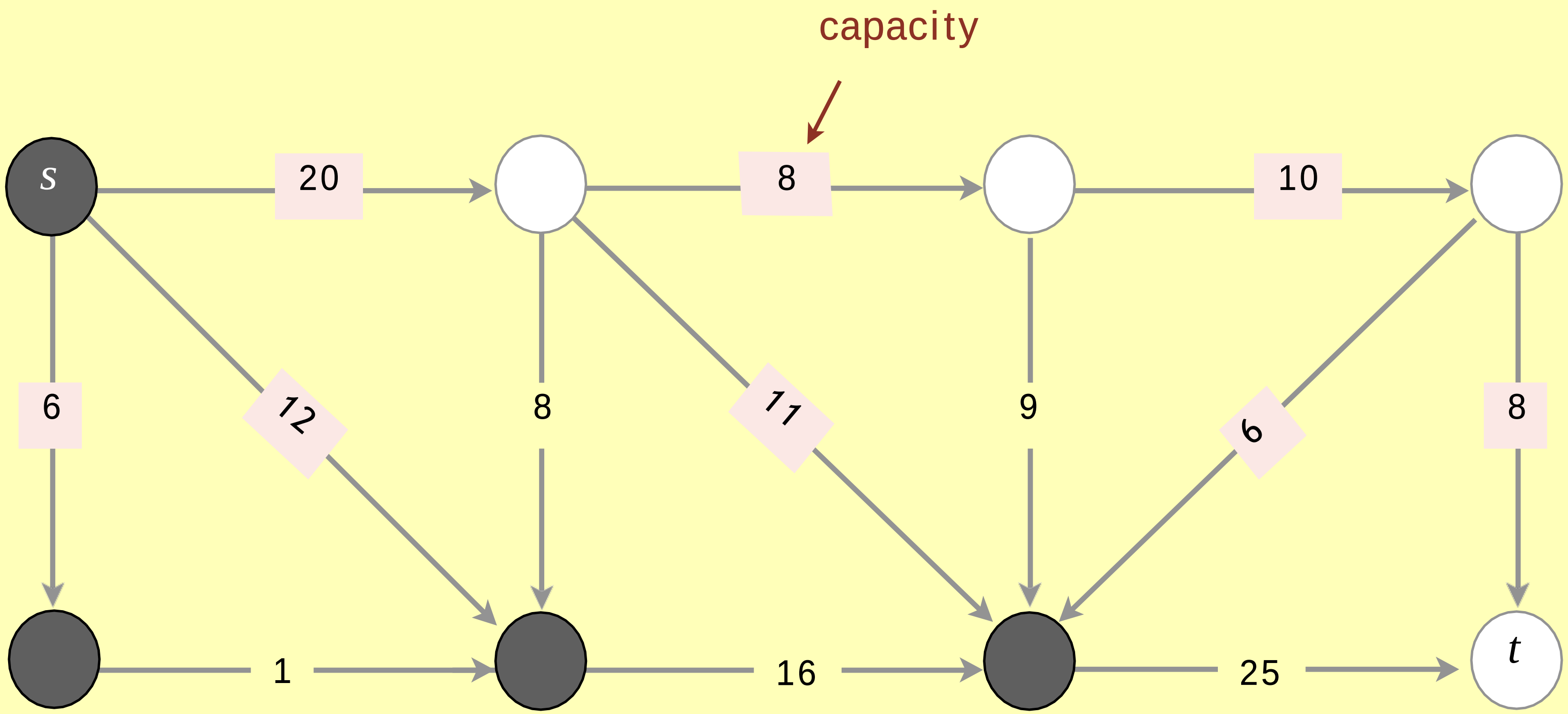
Def. Its capacity is the sum of the capacities of the edges from A to B.

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

Min-cut problem. Find a cut of minimum capacity.



What is the capacity of the cut?



Relationship between flows and cuts

Flow value lemma. Let f be any st-flow and let (A, B) be any st-cut. Then, the value of the st-flow f equals the net flow across the cut (A, B).

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

Proof. We know, $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

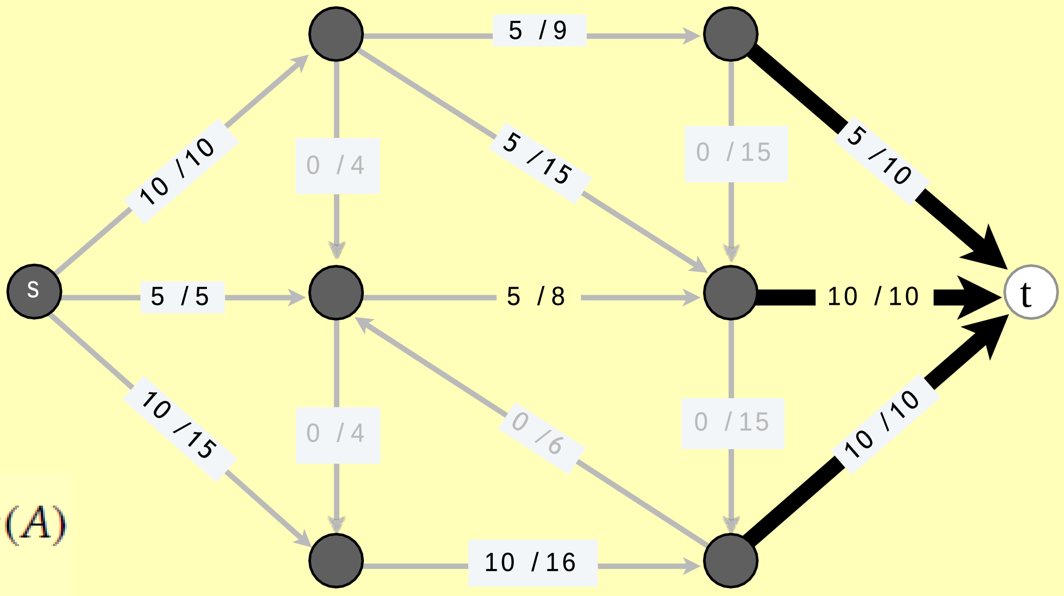
Every node v in A is internal except s, and we know that for all such nodes, $f^{out}(v) = f^{in}(v)$

So,

$$val(f) = \sum_{v \in A} (f^{out}(v) - f^{in}(v))$$

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

net flow across cut = 5 + 10 + 10 = 25



value of flow = 25

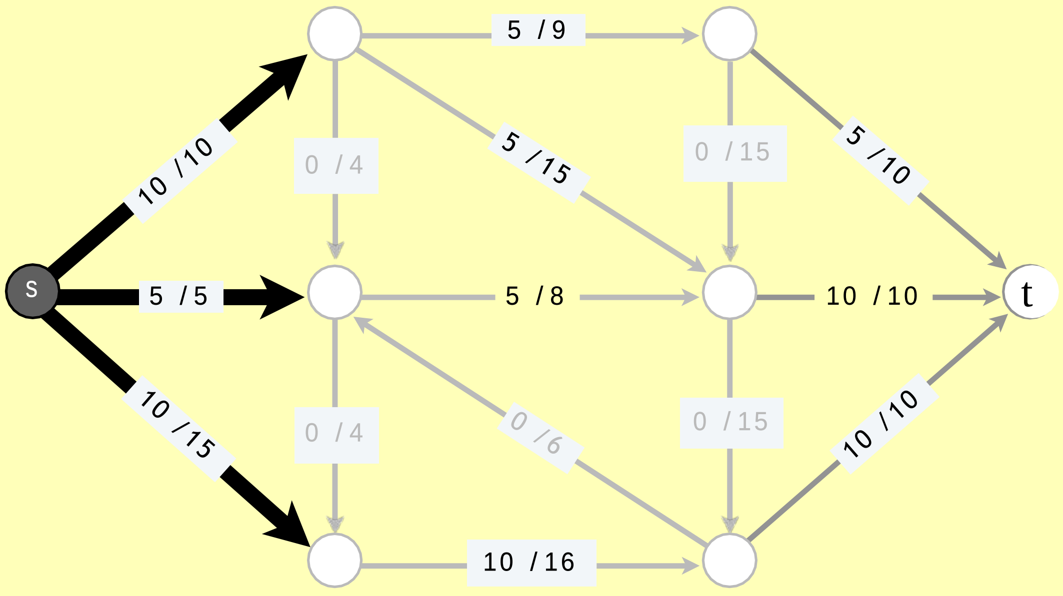
Relationship between flows and cuts

Flow value lemma. Let f be any st-flow and let (A, B) be any st-cut.

Then, the value of the flow f equals the net flow across the cut (A, B).

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

net flow across cut = 10 + 5 + 10 = 25



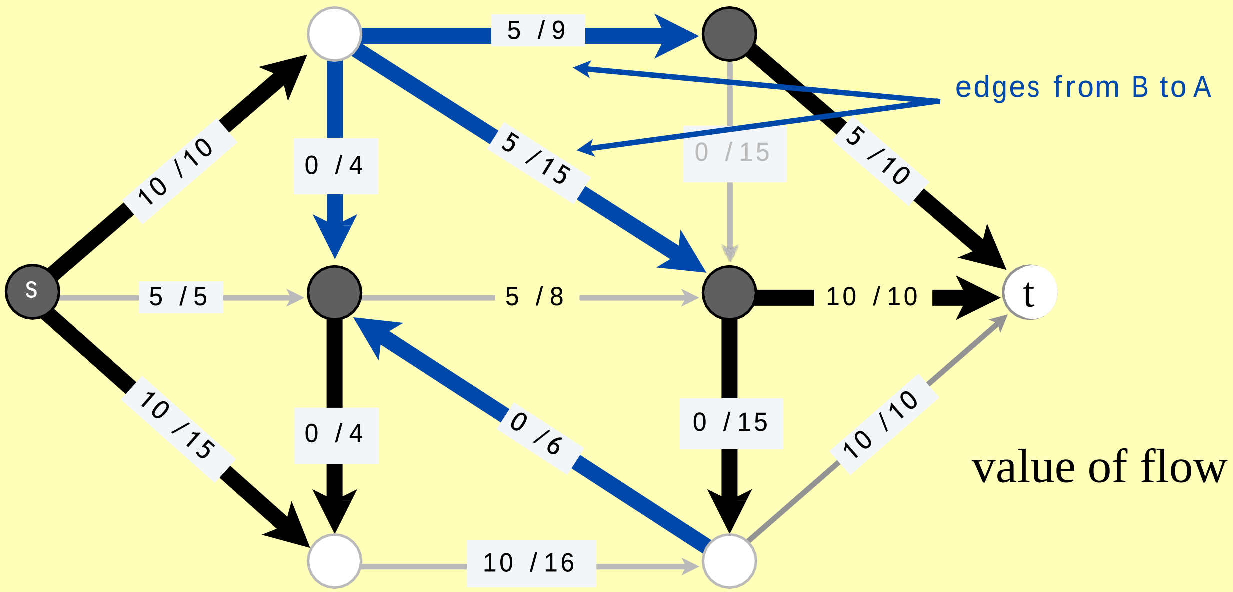
value of flow = 25

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the st-cut (A, B).

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

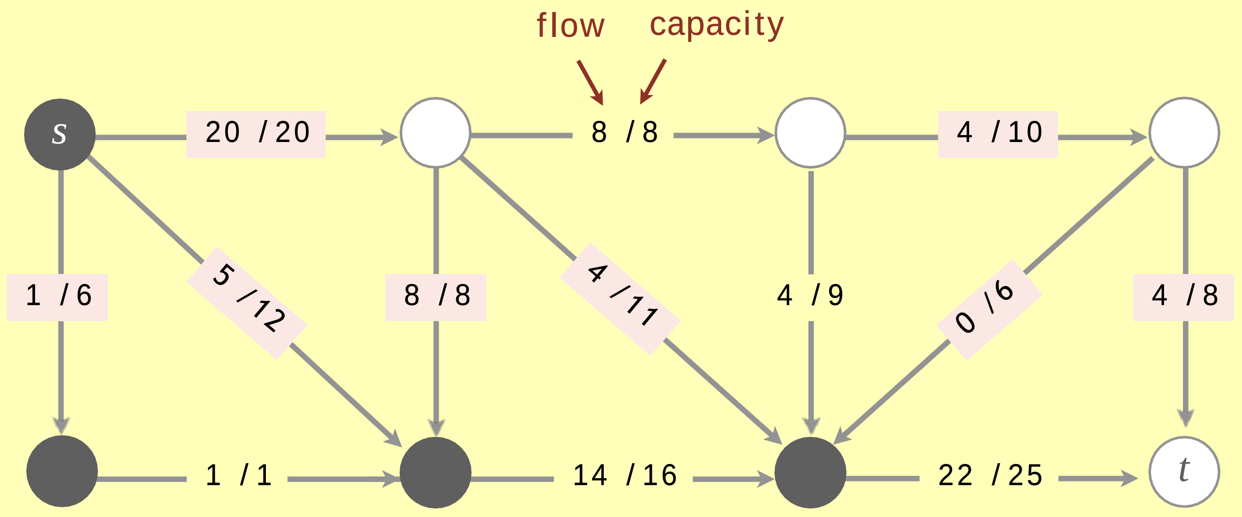
net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) – (5 + 5 + 0 + 0) = 25



value of flow = 25

Relationship between flows and cuts

What is the net flow across the given cut?



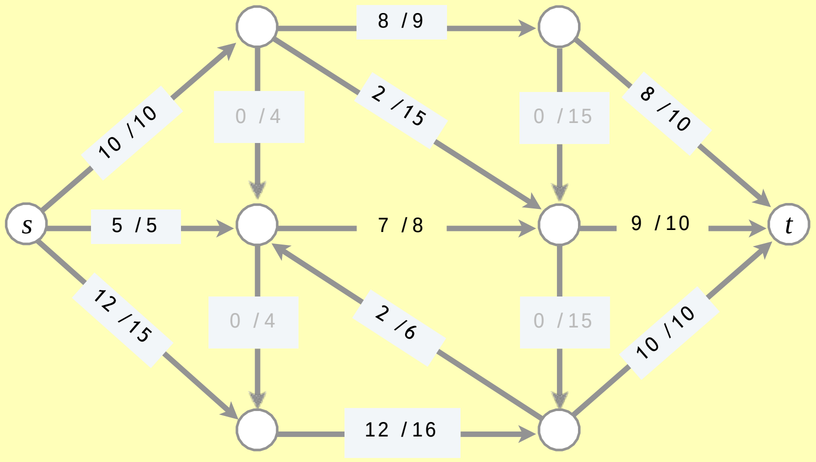
Relationship between flows and cuts

Weak duality. Let f be any st-flow and (A, B) be any st-cut. Then, $val(f) \leq cap(A, B)$.

Pf.

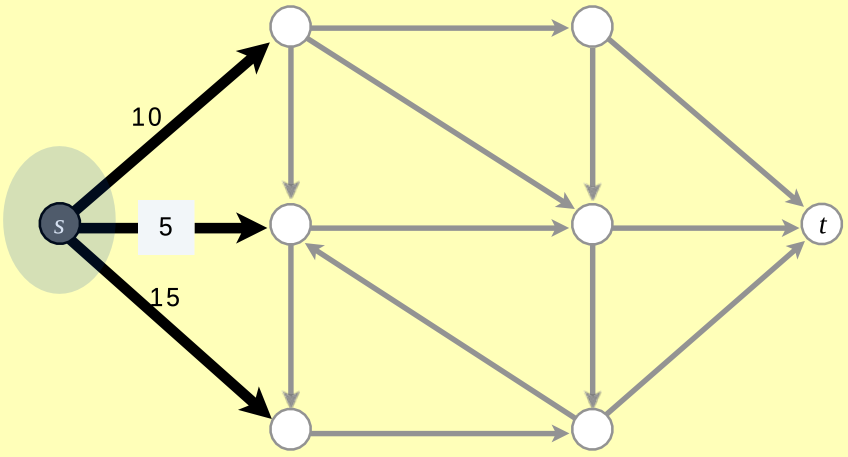
$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&\leq \sum_{e \text{ out of } A} f(e) \\
&\leq \sum_{e \text{ out of } A} c(e) \\
&= cap(A, B) \quad \blacksquare
\end{aligned}
$$

flow value lemma



value of flow = 27

≤



capacity of cut = 30

Relationship between flows and cuts

Theorem 1. If f is an st-flow such that there is no s-t path in the residual graph G_f, then there is an st-cut (A^*, B^*) in G for which $v(f) = c(A^*, B^*)$. Consequently, f has the maximum value of any flow in G, and (A^*, B^*) has the minimum capacity of any st-cut in G.

Pf. The statement claims the existence of the cut (A^*, B^*) satisfying a certain desirable property;

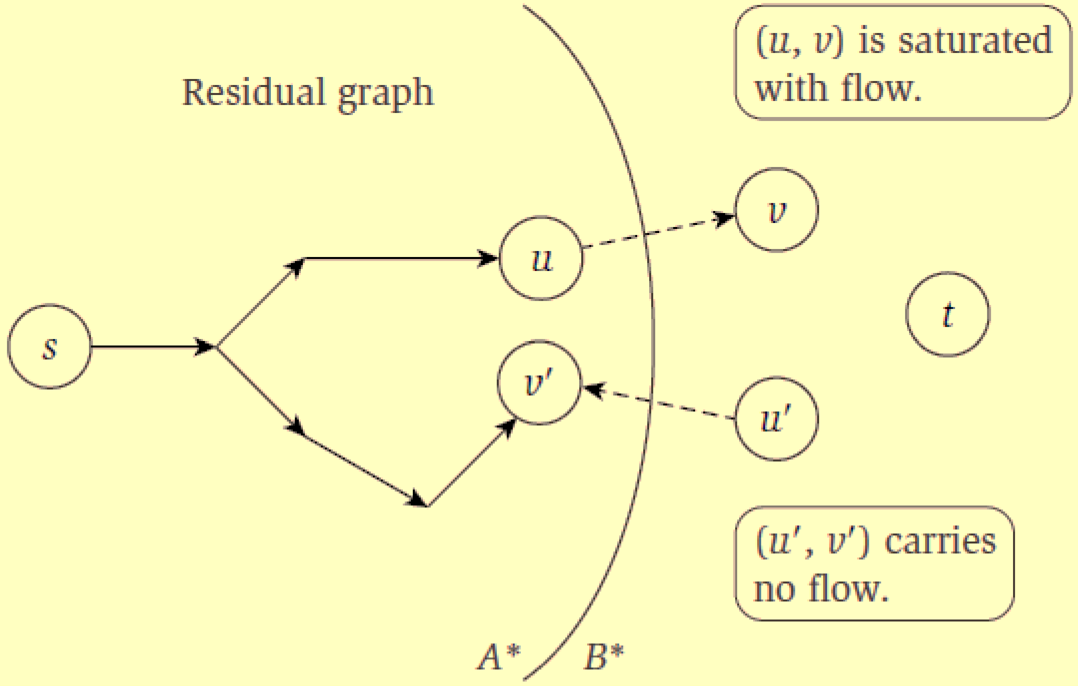
- To prove, we must identify such a cut.
- Let A^* denote the set of all nodes v in G for which there is an s-v path in the final G_f. Let B^* denote the set of all other nodes: $B^* = V - A^*$.
- First we establish that (A^*, B^*) is an s-t cut.
- It is clearly a partition of V. The source s belongs to A^* since there is always a path from s to s.
- Moreover, $t \notin A^*$ by the assumption that there is no s-t path in G_f; hence $t \in B^*$ as desired.

Relationship between flows and cuts

Theorem 1. If f is an st-flow such that there is no s-t path in the residual graph G_f, then there is an st-cut (A^*, B^*) in G for which $v(f) = c(A^*, B^*)$. Consequently, f has the maximum value of any flow in G, and (A^*, B^*) has the minimum capacity of any st-cut in G.

Pf. Suppose that $e = (u, v)$ is an edge in G for which $u \in A^*$ and $v \in B^*$. We claim that $f(e) = c(e)$.

- For if not, e would be a forward edge in the residual graph G_f, and there will be a path s-u-v in G_f contradicting our assumption that v $\in B^*$.
- So all edges out of A^* are completely saturated with flow.

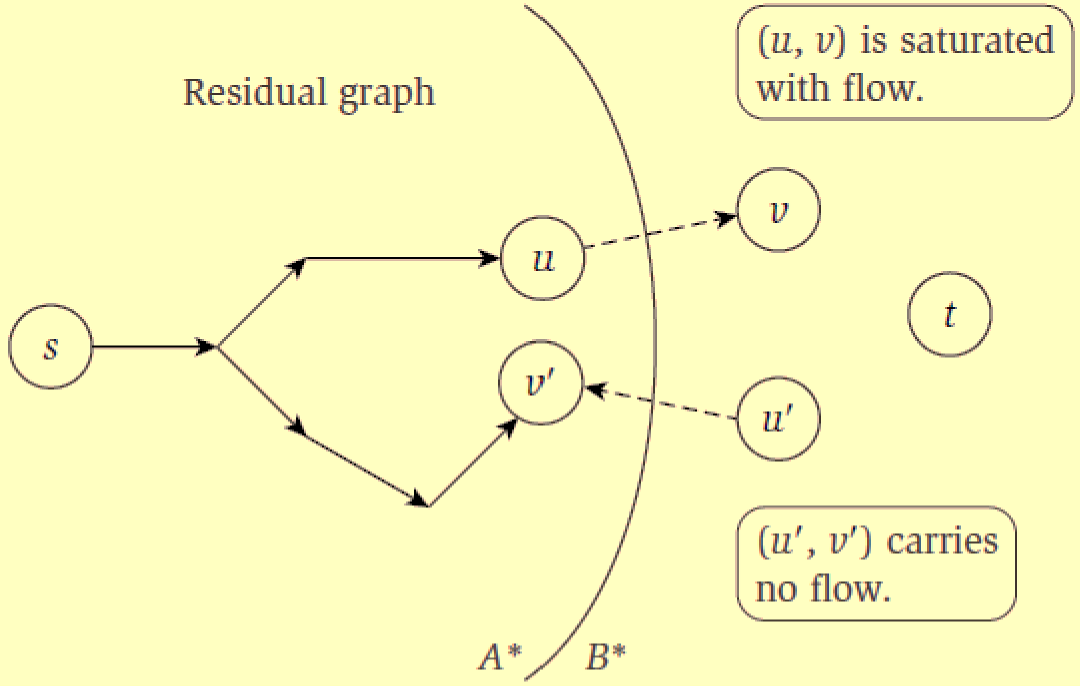


Relationship between flows and cuts

Theorem 1. **If f is an st-flow such that there is no s-t path in the residual graph G_f, then there is an st-cut (A^*, B^*) in G for which $v(f) = c\,(A^*, B^*)$. Consequently, f has the maximum value of any flow in G, and (A^*, B^*) has the minimum capacity of any st-cut in G.**

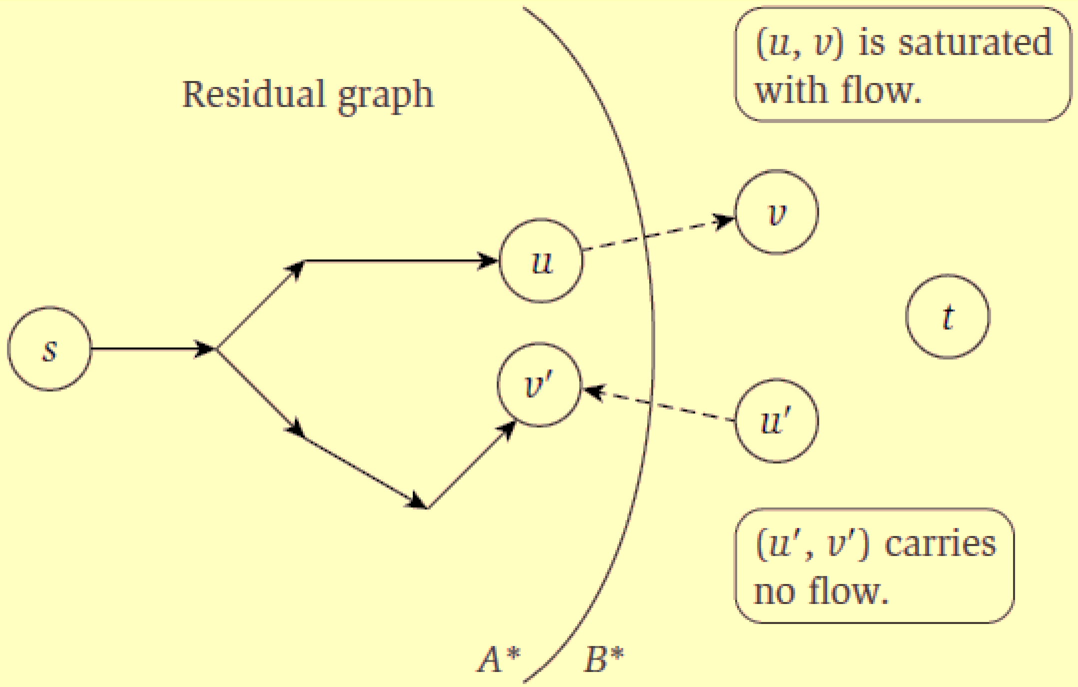
Pf. Suppose that $e' = (u', v')$ is an edge in G for which $v' \in A^*$ and $u' \in B^*$. We claim that $f(e') = 0$.

- For if not, e' would produce a backward edge $e'' = (v', u')$ in the residual graph G_f, and there will be a path s-v'-u' in G_f contradicting our assumption that $u' \in B^*$.
- All edges into A^* are completely unused.



Relationship between flows and cuts

Theorem 1. **If f is an st-flow such that there is no s-t path in the residual graph G_f, then there is an st-cut (A^*, B^*) in G for which $v(f) = c(A^*, B^*)$. Consequently, f has the maximum value of any flow in G, and (A^*, B^*) has the minimum capacity of any st-cut in**



Pf.

$$
\begin{aligned}
\nu(f) &= f^{\text{out}}(A^*) - f^{\text{in}}(A^*) \\
&= \sum_{e \text{ out of } A^*} f(e) - \sum_{e \text{ into } A^*} f(e) \\
&= \sum_{e \text{ out of } A^*} c_e - 0 \\
&= c(A^*, B^*). \quad \blacksquare
\end{aligned}
$$

Max-flow min-cut theorem

Augmenting path theorem. A flow f is a max flow iff. no augmenting paths.

Max-flow min-cut theorem. In every flow network, the maximum value of an *st*-flow is equal to the minimum capacity of an *st*-cut.

Pf. The point is that f in Theorem 1 must be a maximum *st*-flow;

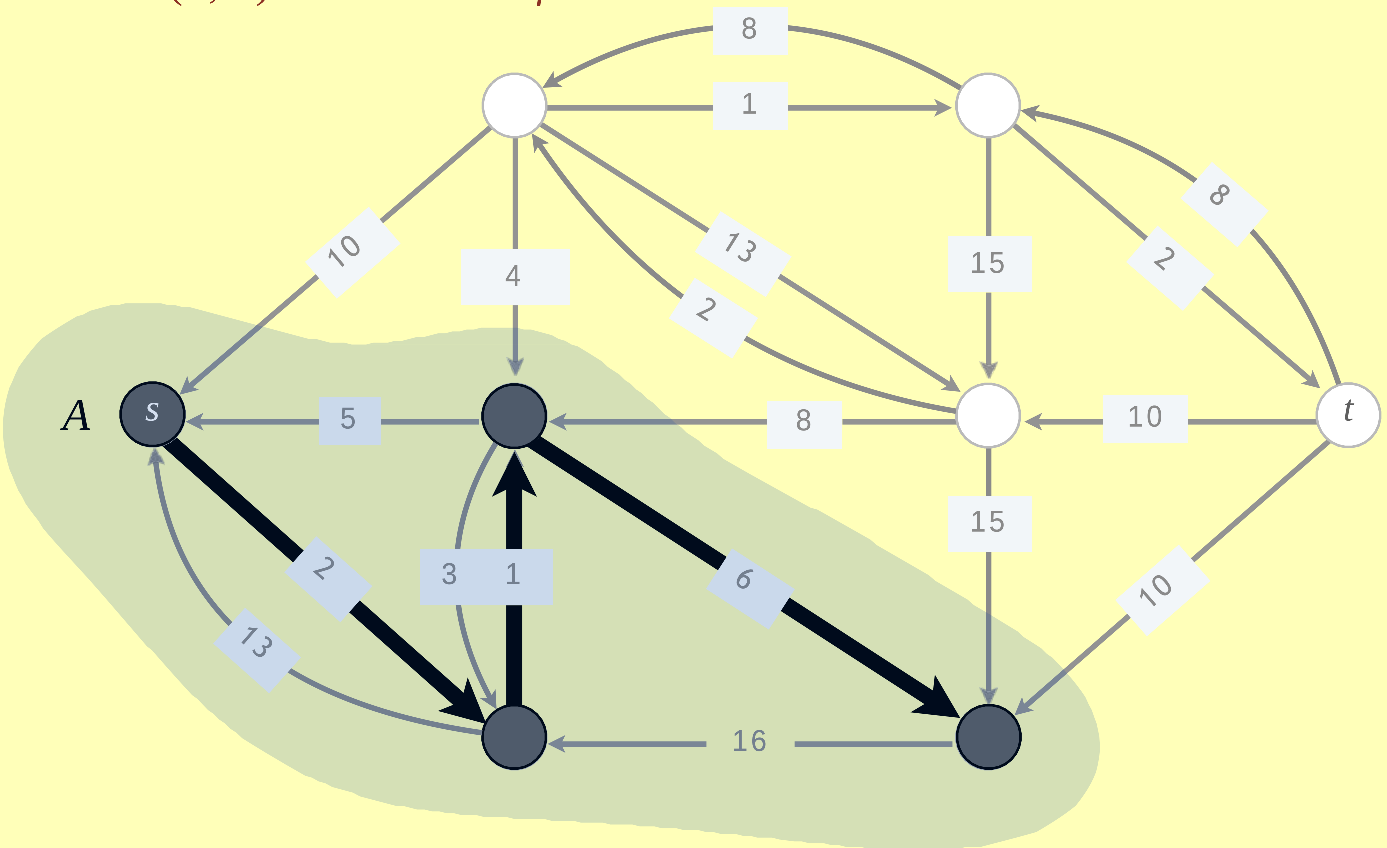
- If there were a flow $f' > f$, the value of f' would exceed the capacity of (A, B), and this would contradict the weak duality.
- Similarly, it follows that (A, B) in Theorem 1 is a minimum cut and no other cut can have smaller capacity.
- If there were a cut (A', B') of smaller capacity, it would be less than the value of f, and this again would contradict the weak duality.
- Due to these implications, the Max-Flow Min-Cut Theorem holds its claim.

Computing a minimum cut from a maximum flow

Theorem. Given any max flow f, can compute a min cut (A, B) in $O(m)$ time (m = no. of edges in the residual graph).

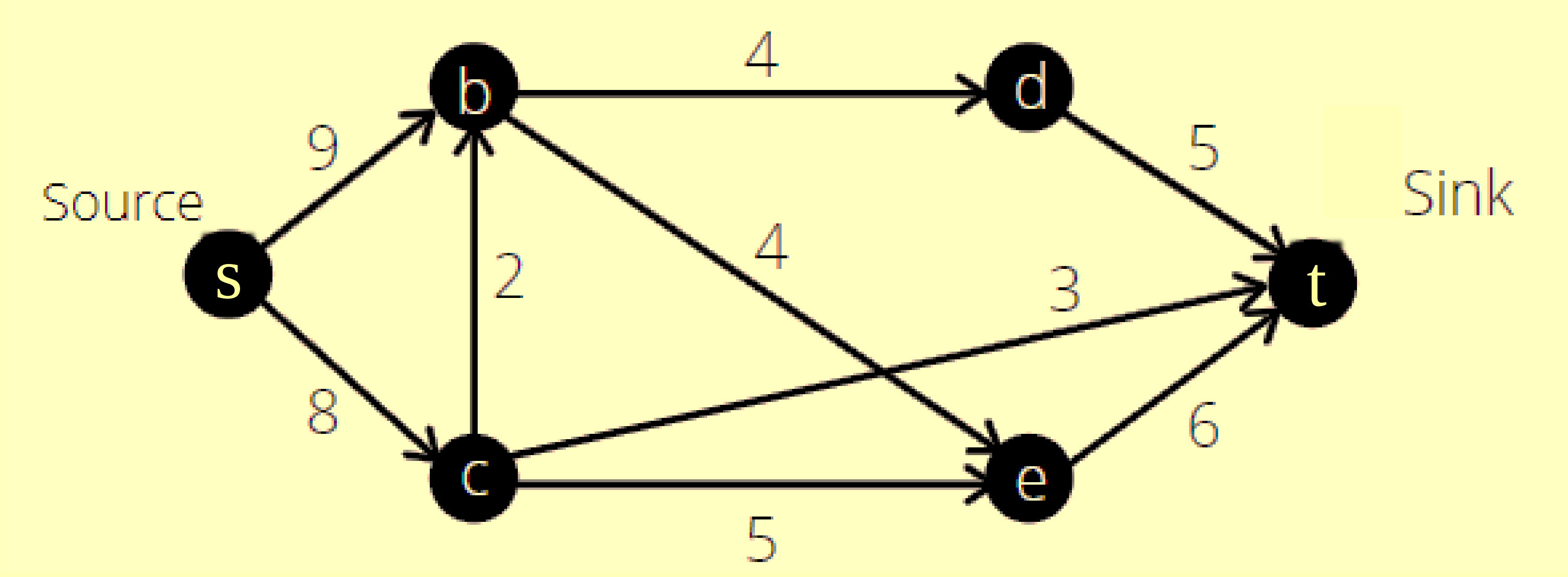
Pf. Let A = set of nodes reachable from s in residual network G_f using BFS/DFS. ▪

argument from previous slide implies that capacity of (A, B) = value of flow f



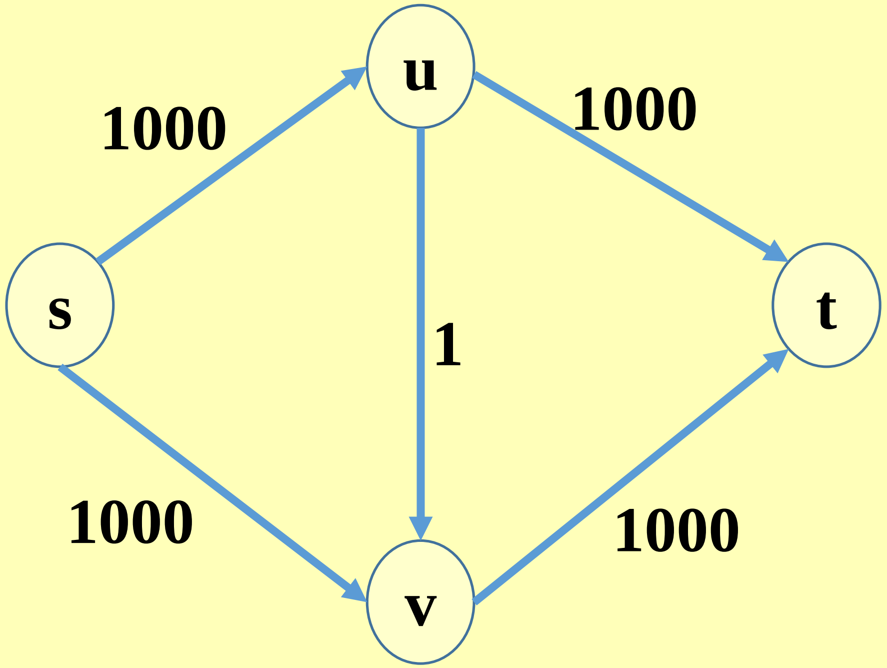
Example

Find the maximum flow and minimum cut capacity in the given graph, identify the cut.



Choosing good augmenting path

- The worst case time complexity of Ford-Fulkerson: $O(m*C)$
- It is a reasonable bound for smaller C value;
- However, it is very weak when C is large.
- For example, if Ford-Fulkerson algorithm selects the augmenting paths in the following order:
  - *s-u-v-t*
  - *s-v-u-t*
  - *s-u-v-t*
  - *s-v-u-t*
  - …….
- The running time will be significantly high.



Choosing good augmenting path: History

year	method	# augmentations	running time
1955	**augmenting path**	$n\,C$	$O(m\,n\,C)$
1972	**fattest path**	$m \log (mC)$	$O(m^2 \log n \log (mC))$
1972	**capacity scaling**	$m \log C$	$O(m^2 \log C)$
1985	**improved capacity scaling**	$m \log C$	$O(m\,n \log C)$
1970	**shortest augmenting path**	$m\,n$	$O(m^2\,n)$
1970	**level graph**	$m\,n$	$O(m\,n^2)$
1983	**dynamic trees**	$m\,n$	$O(m\,n \log n)$

fat paths

shortest paths

Max-flow algorithms: History

year	method	worst case	discovered by
1951	simplex	$O(m\, n^2\, C)$	Dantzig
1955	augmenting paths	$O(m\, n\, C)$	Ford–Fulkerson
1970	shortest augmenting paths	$O(m\, n^2)$	Edmonds–Karp, Dinitz
1974	blocking flows	$O(n^3)$	Karzanov
1983	dynamic trees	$O(m\, n \log n)$	Sleator–Tarjan
1985	improved capacity scaling	$O(m\, n \log C)$	Gabow
1988	push-relabel	$O(m\, n \log (n^2 / m))$	Goldberg–Tarjan
1998	binary blocking flows	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	compact networks	$O(m\, n)$	Orlin
2014	interior-point methods	$\tilde{O}(m\, n^{1/2} \log C)$	Lee–Sidford
2016	electrical flows	$\tilde{O}(m^{10/7}\, C^{1/7})$	Mądry
20xx		???	

max-flow algorithms with m edges, n nodes, and integer capacities between 1 and C

Choosing good augmenting path

- A natural idea is to select the path with largest bottleneck capacity.
- However, to find such paths can slow down each individual iteration by quite a bit.
- We will avoid this slowdown by not worrying about selecting the path that has exactly the largest bottleneck capacity. Instead, we will maintain a scaling parameter Δ, and we will look for paths that have bottleneck capacity of at least Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph G_f consisting only of edges with residual capacity of at least Δ.
- We will work with values of Δ that are powers of 2 and not greater than C.

Scaling-Max-Flow Algorithm

Scaling-Max-Flow(G, s, t, c)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$\Delta \leftarrow$ largest power of 2 $\leq C$.

WHILE ($\Delta \geq 1$)

  $G_f(\Delta) \leftarrow \Delta$-residual network of G with respect to flow f.

  WHILE (there exists an $s \leadsto t$ path P in $G_f(\Delta)$)

    $f \leftarrow$ AUGMENT(f, c, P).

    Update $G_f(\Delta)$.

  $\Delta \leftarrow \Delta / 2$.

Δ-scaling phase

RETURN f.

Scaling-Max-Flow Algorithm: Analysis

- Assumption. All edge capacities are integers between 1 and C.
- Lemma 1. There are $1 + \lfloor \log C \rfloor$ scaling phases.
- Pf. Initially $C / 2 < \Delta \leq C$; Δ decreases by a factor of 2 in each iteration. So, total iterations will be $1 + \lfloor \log C \rfloor$.
- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\leq$ val(f) + m Δ.

Pf. At the end of a Δ-scaling phase, lets identify a cut (A, B) where A denotes the set of all nodes v in G for which there is an s-v path in $G_f(\Delta)$ and B denotes the set of all other nodes: $B = V - A$.

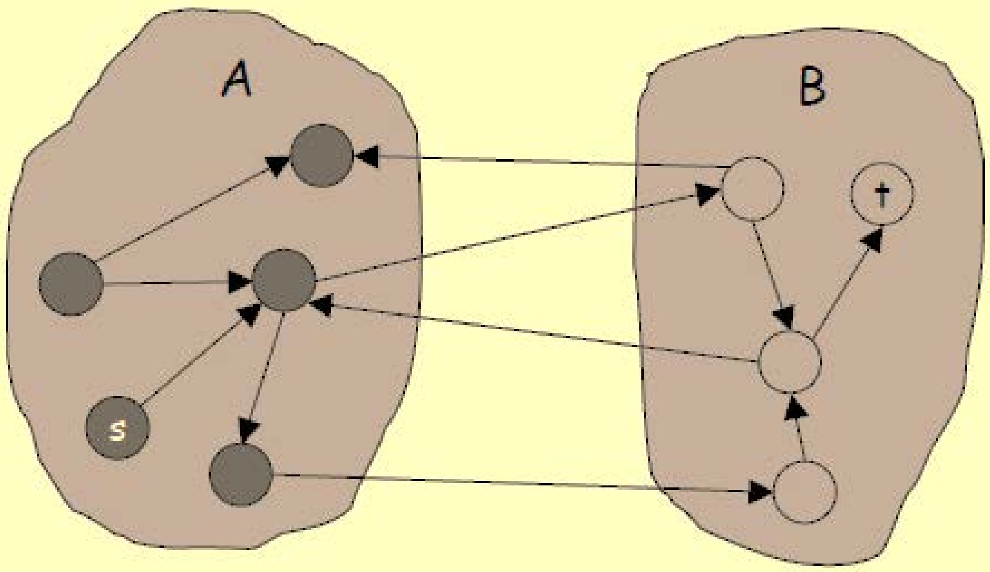
- We can observe that (A, B) is indeed an st-cut as otherwise the phase would not have ended.

Scaling-Max-Flow Algorithm: Analysis

- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\leq$ val(f) + m Δ.

Pf. We can easily realize that all edges e out of A are almost saturated and they satisfy $c_e < f(e) + \Delta$ and all edges into A are almost empty and they satisfy f (e) < Δ.

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) \\
&\geq \sum_{e \text{ out of } A} (c_e - \Delta) - \sum_{e \text{ into } A} \Delta \\
&= \sum_{e \text{ out of } A} c_e - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ into } A} \Delta \\
&\geq c(A, B) - m\Delta.
\end{aligned}
$$



So, $c(A, B) \leq v(f) + m\Delta \quad \Rightarrow$ max-flow $\leq v(f) + m\Delta$

Scaling-Max-Flow Algorithm: Analysis

- Lemma 3. The number of augmentations in a scaling phase is at most $2m$.

Pf. The statement is clearly true in the first scaling phase: we can use each of the edges out of s only for at most one augmentation in that phase.

- Now consider a later scaling phase Δ, and let f_p be the flow at the end of the previous scaling phase. In that phase, we used $\Delta' = 2\Delta$ as our scaling parameter.
- By Lemma 2, the maximum flow f^* is at most $v(f^*) \leq v(f_p) + m\Delta' = v(f_p) + 2m\Delta$.
- In the Δ-scaling phase, each augmentation increases the flow by at least Δ, and hence there can be at most 2m augmentations..

Scaling-Max-Flow Algorithm: Analysis

- Lemma 4. The Scaling Max-Flow Algorithm in a graph with m edges and integer capacities finds a maximum flow in at most 2m(1+ $\lfloor \log C \rfloor$) augmentations. It can be implemented to run in at most $O(m^2 \log C)$ time.

Pf. Lemma 1 + Lemma 3 ⇒ O(m log C) augmentations.

- Finding an augmenting path takes O(m) time.
- So, the running time of the Scaling Max-Flow Algorithm is $O(m^2 \log C)$.

Scaling-Max-Flow(G, s, t, c)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$\Delta \leftarrow$ largest power of 2 $\leq C$.

WHILE ($\Delta \geq 1$)

  $G_f(\Delta) \leftarrow \Delta$-residual network of G with respect to flow f.

  WHILE (there exists an $s \rightsquigarrow t$ path P in $G_f(\Delta)$)

    $f \leftarrow$ AUGMENT(f, c, P).

    Update $G_f(\Delta)$.

  $\Delta \leftarrow \Delta / 2$.

Δ-scaling phase

RETURN f.

Max-flow and min-cut applications

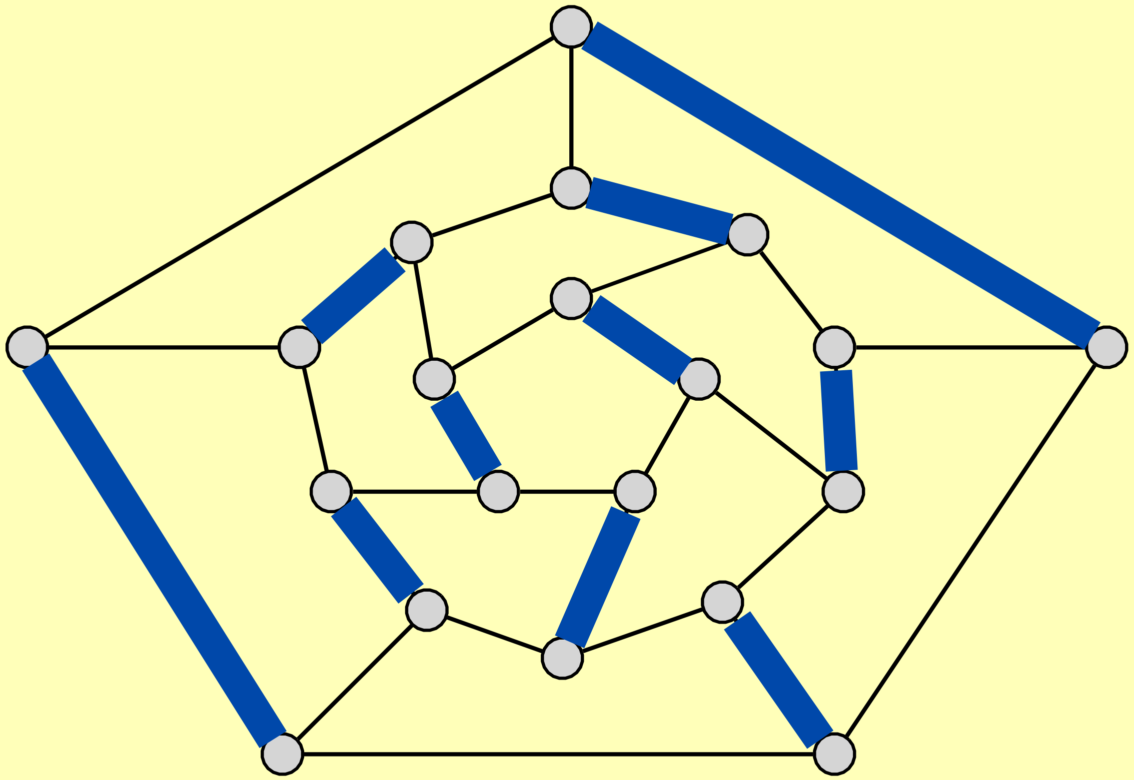
Max-flow and min-cut problems are widely applicable model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Markov random fields.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.

Matching

Def. Given an undirected graph $G = (V, E)$, subset of edges $M \subseteq E$ is a matching if each node appears in at most one edge in M.

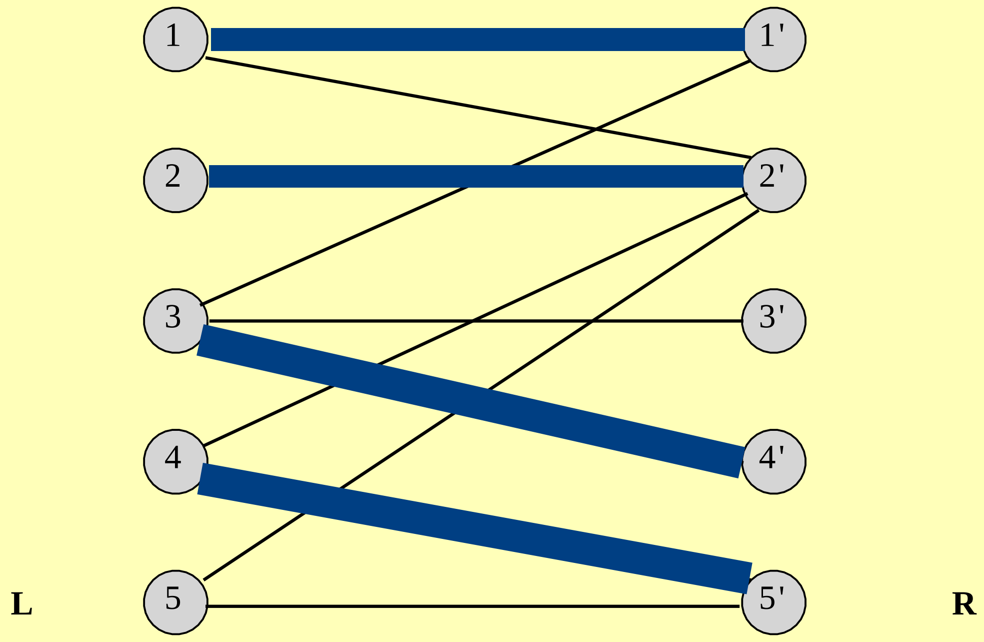
Max matching. Given a graph G, find a max-cardinality matching.



Bipartite matching

Def. A graph G is bipartite if the nodes can be partitioned into two subsets L and R such that every edge connects a node in L with a node in R.

Bipartite matching. Given a bipartite graph $G = (L \cup R, E)$, find a max-cardinality matching.

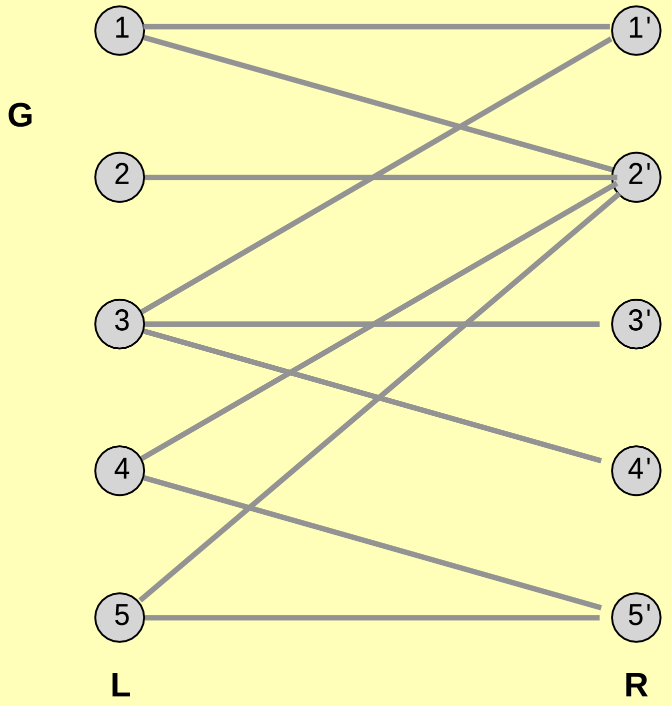


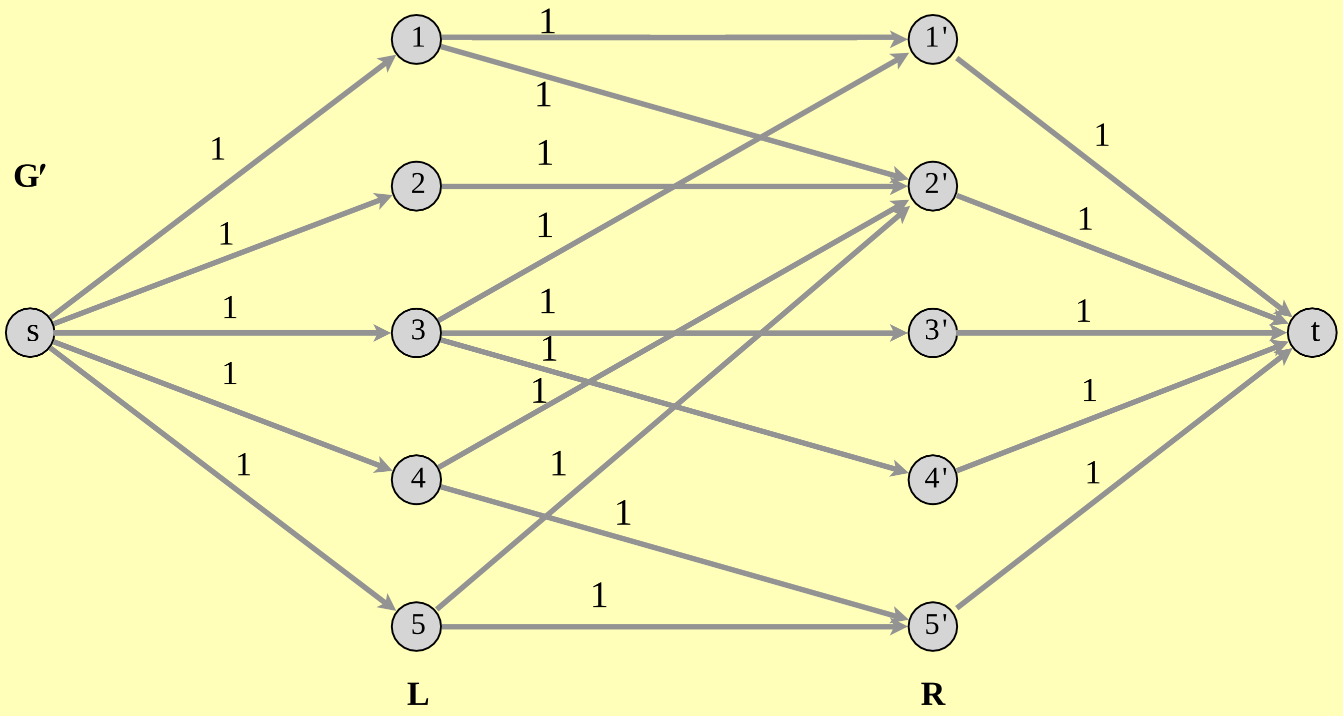
matching: 1-1', 2-2', 3-4', 4-5'

Bipartite matching: max-flow formulation

Formulation.

- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$ from the given bipartite graph $G = (L \cup R, E)$.
- Direct all edges from L to R, and assign the capacity as 1 for each.
- Add unit-capacity edges from s to each node in L.
- Add unit-capacity edges from each node in R to t.
- Now, run the Ford-Fulkerson Algorithm and find the max flow value which gives the maximum cardinality of bipartite matching.





Max-flow formulation: proof of correctness

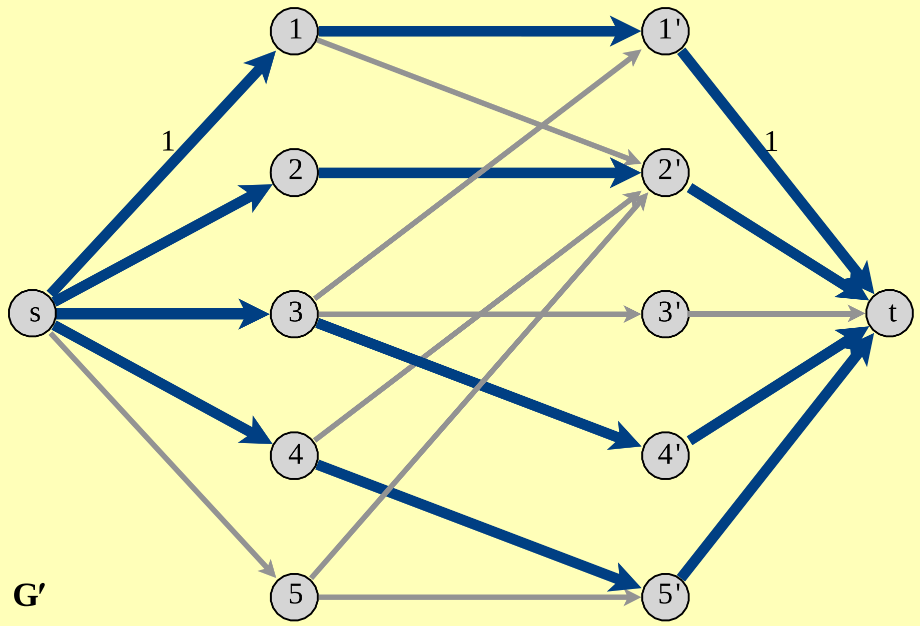
- Suppose there is a matching in G consisting of k edges of the form (x, y) where $x \in L$ and y $\in R$.
- Consider the flow f that sends one unit along each path of the form s-x-y-t, i.e., $f(e) = 1$ for each edge on these paths. We can verify easily that the capacity and conservation conditions are indeed met and that f is an s-t flow of value k.
- Conversely, suppose there is a flow f' in G' of value k. We know there is an integer-valued flow f of value k; and since all capacities are 1, this means that $f(e)$ is equal to either 0 or 1 for each edge e.
- Now, consider the set M' of edges of the form (x, y) on which the flow value is 1.

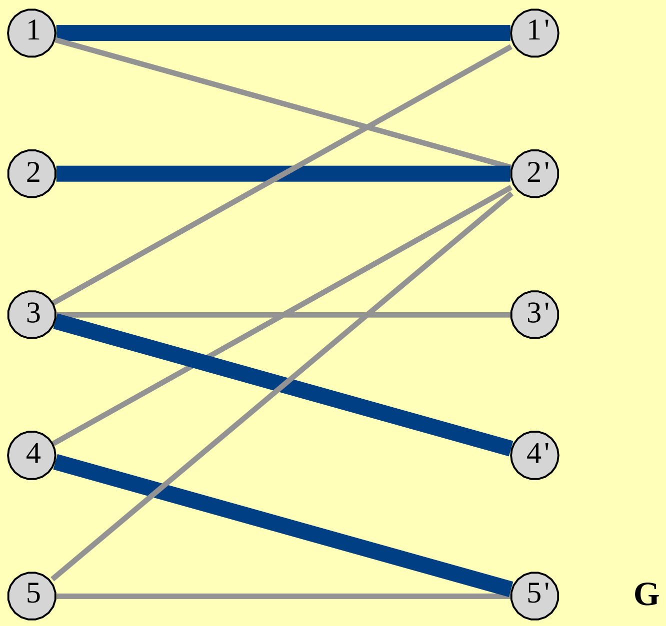
Max-flow formulation: proof of correctness

Observation 1. *M'* contains *k* edges where *k* is the max flow.

Pf. To prove this, consider the cut (*A*, *B*) in *G'* with *A*= {*s*} ∪ *L*.

- The value of the flow is the total flow leaving *A*, minus the total flow entering A.
- The first of these terms is simply the cardinality of *M'* since these are the edges leaving *A* that carry flow, and each carries exactly one unit of flow.
- The second of these terms is 0, since there are no edges entering *A*.
- Thus, *M'* contains *k* edges.



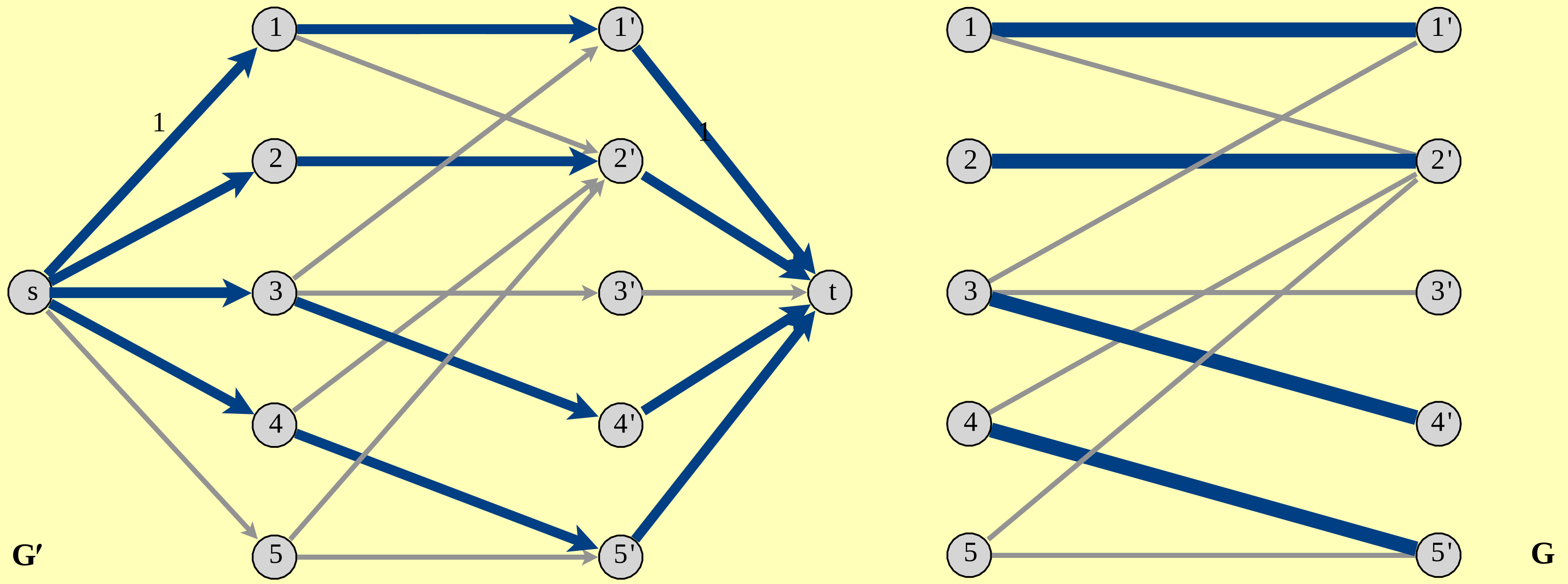


Max-flow formulation: proof of correctness

Observation 2. Each node in L is the tail of at most one edge in M'.

Pf. To prove this, suppose $x \in L$ were the tail of at least two edges in M'.

- Since our flow is integer-valued, this means that at least two units of flow leave from x.
- By conservation of flow, at least two units of flow would have to come into x—but this is not possible, since only a single edge of capacity 1 enters x.
- Thus x is the tail of at most one edge in M'.

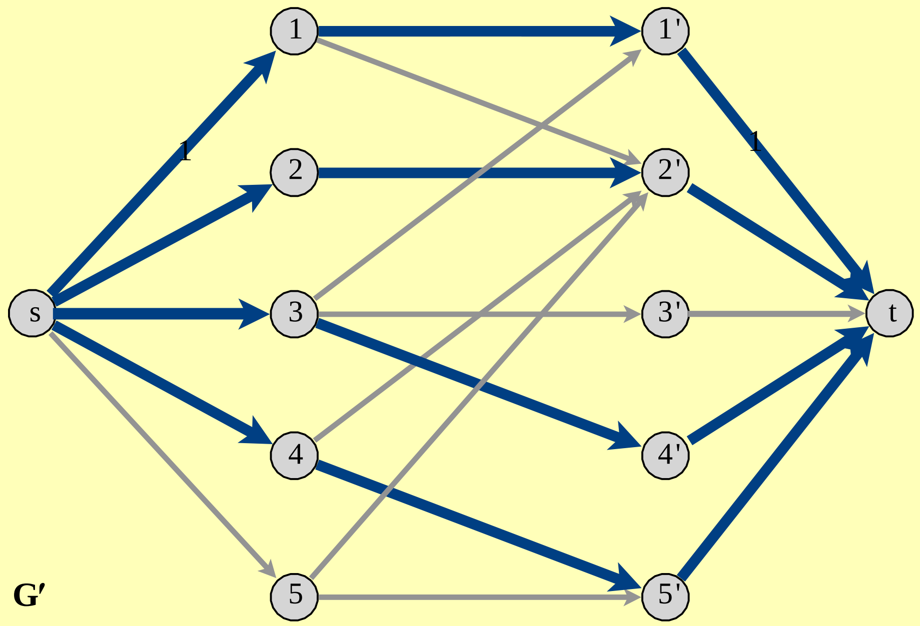


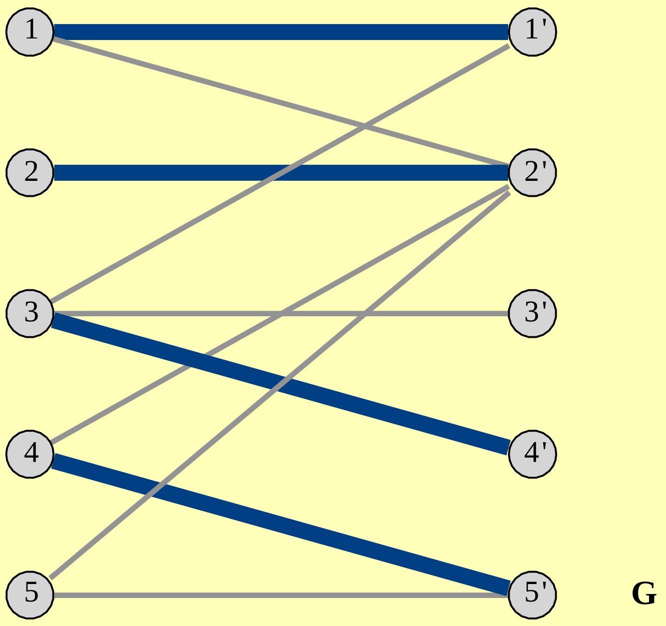
Max-flow formulation: proof of correctness

Observation 3. Each node in R is the head of at most one edge in M'.

Pf. By the same reasoning as previous.

Conclusion. The size of the maximum matching in G is equal to the value of the maximum flow in G'. and the edges in such a matching in G are the edges that carry flow from L to R in G'.





Network flow II: quiz

What is running time of Ford–Fulkerson algorithms to find a max- cardinality matching in a bipartite graph with $|L| = |R| = n$?

A. $O(m + n)$

B. $O(mn)$

C. $O(mn^2)$

D. $O(m^2n)$

Perfect matchings in bipartite graphs

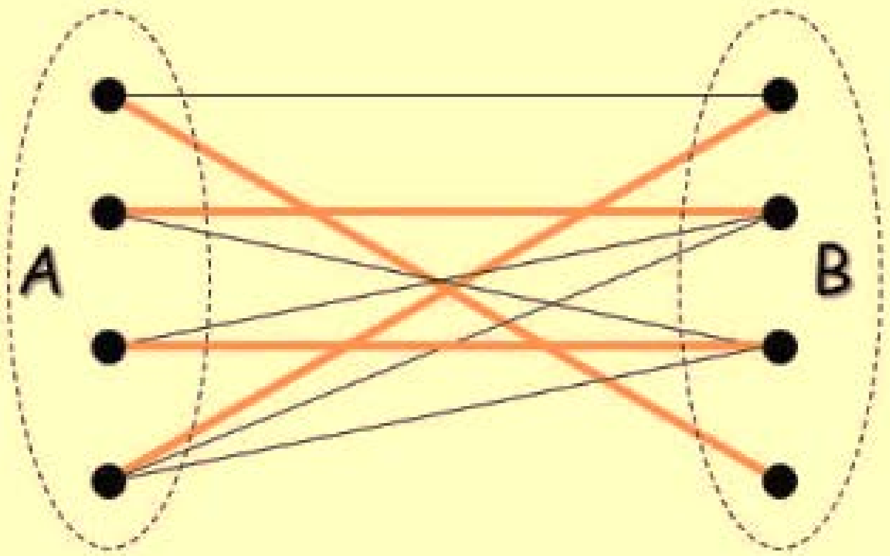
Def. Given a graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a perfect matching if each node of G appears in exactly one edge in M.

- Condition for perfect matching: $|L| = |R| = |M|$

Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

- Clearly, we must have $|L| = |R|$.



Perfect matchings in bipartite graphs

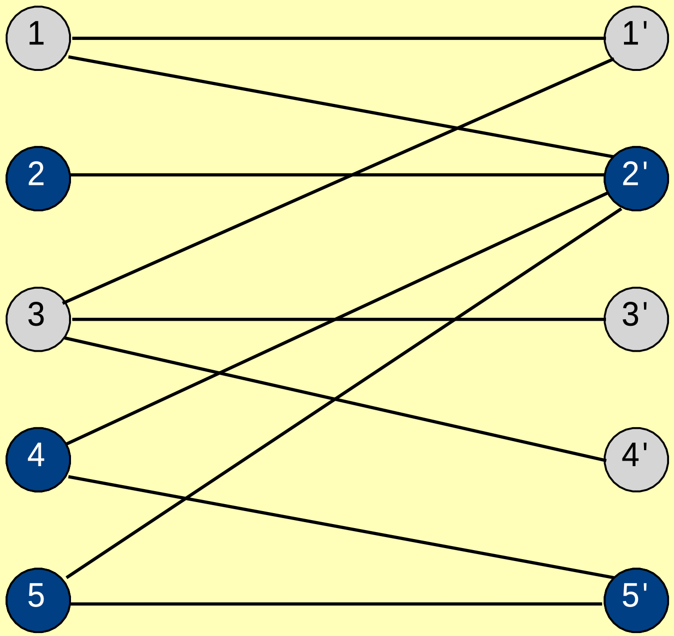
Notation. Let S be a subset of nodes, and let $N(S)$ be the set of nodes adjacent to nodes in S.

Observation. If a bipartite graph $G = (L \cup R, E)$ has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. Each node in S has to be matched to a different node in $N(S)$. ▪

$S = \{2, 4, 5\}$

$N(S) = \{2', 5'\}$



no perfect matching