Design and Analysis of Algorithms
Network Flow

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

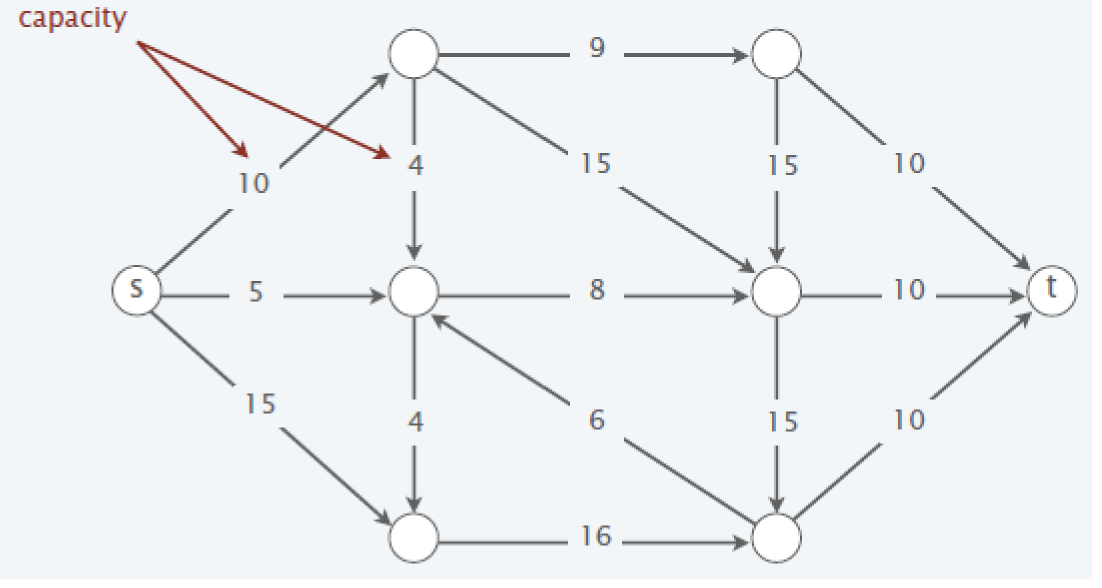
- Max-Flow and Min-Cut Problems
- Ford-Fulkerson Algorithm
- Max-Flow Min-Cut Theorem
- Capacity-Scaling Algorithm
- Shortest Augmenting Paths
- Blocking-Flow Algorithm

Flow Network

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

- Digraph (V, E) with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

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

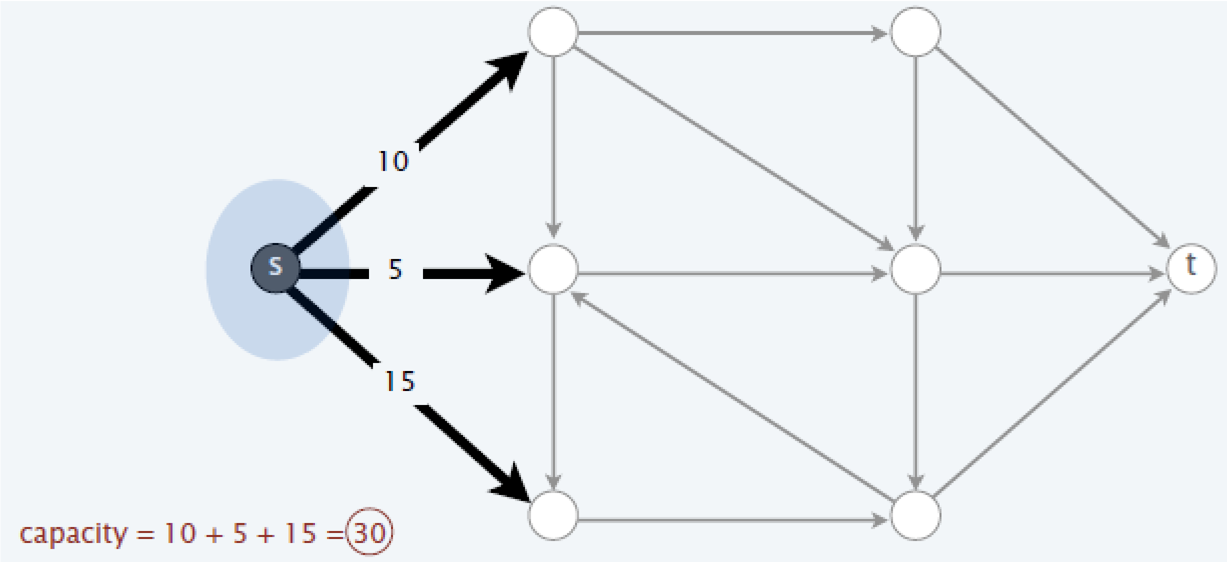


Minimum-Cut Problem

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

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

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

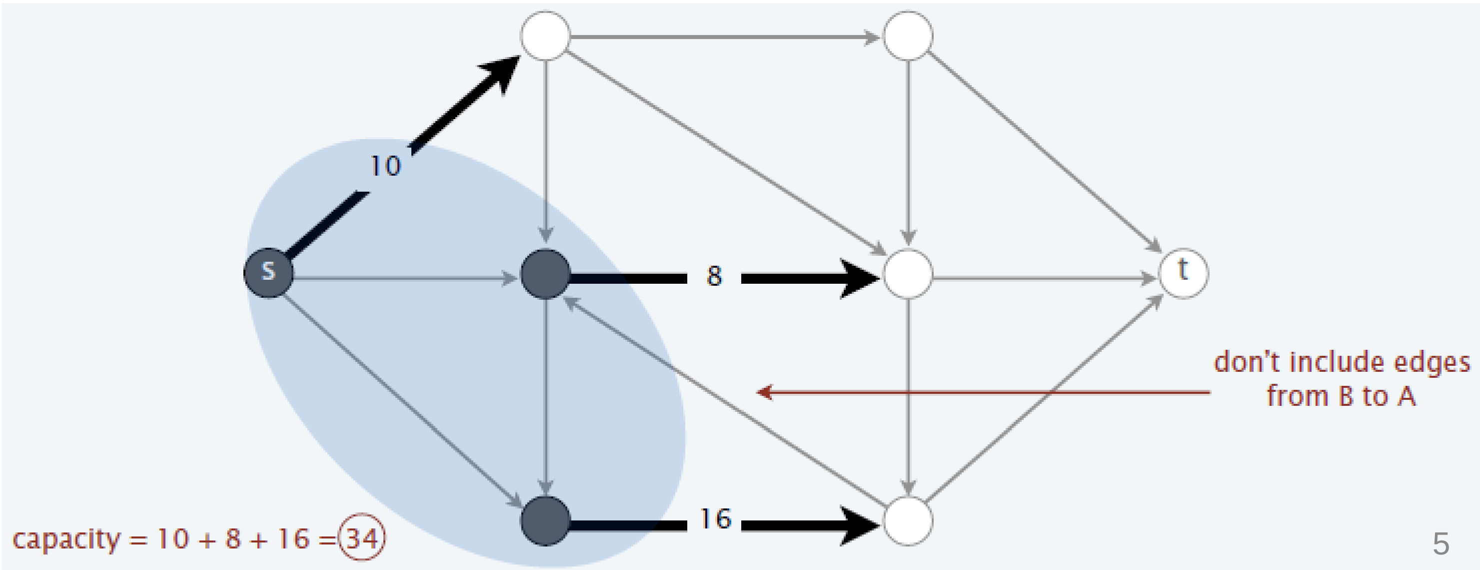


Minimum-Cut Problem

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

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

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



Minimum-Cut Problem

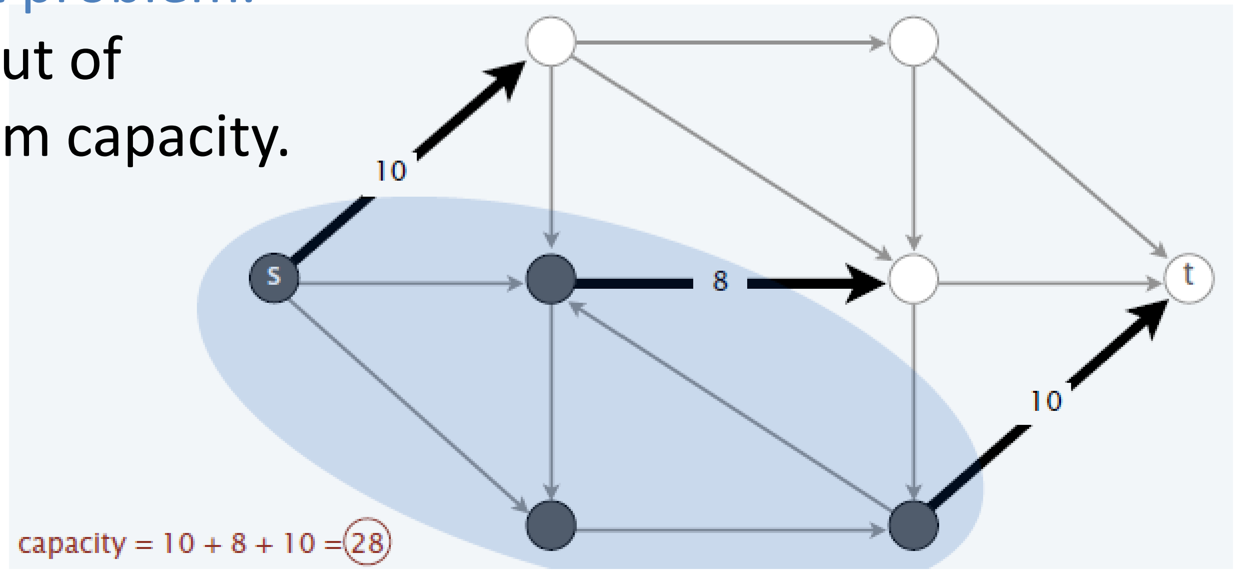
Def. An st-cut (cut) is a partition (A, B) of the vertices with $s \in A$ and $t \in B$.

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.

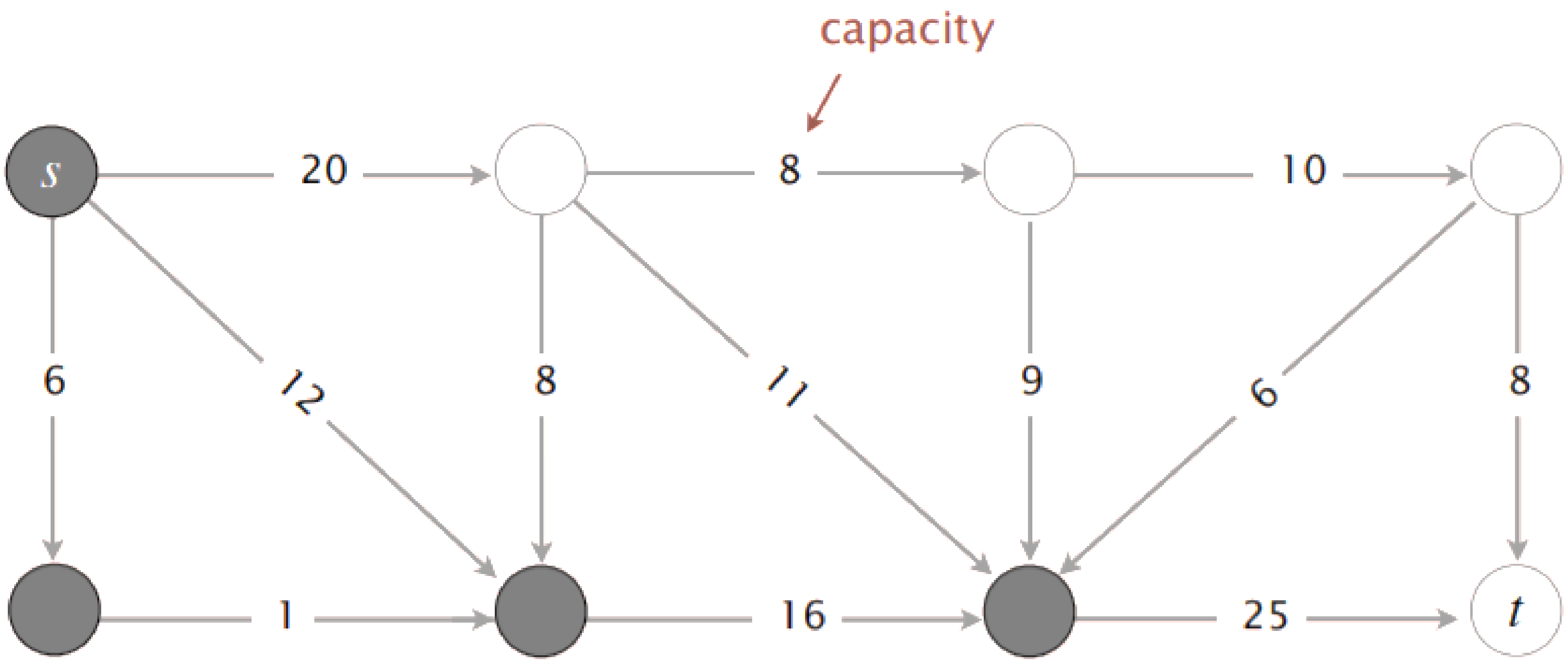
Find a cut of minimum capacity.



Minimum-Cut Problem

What is the capacity of the given *st*-cut?

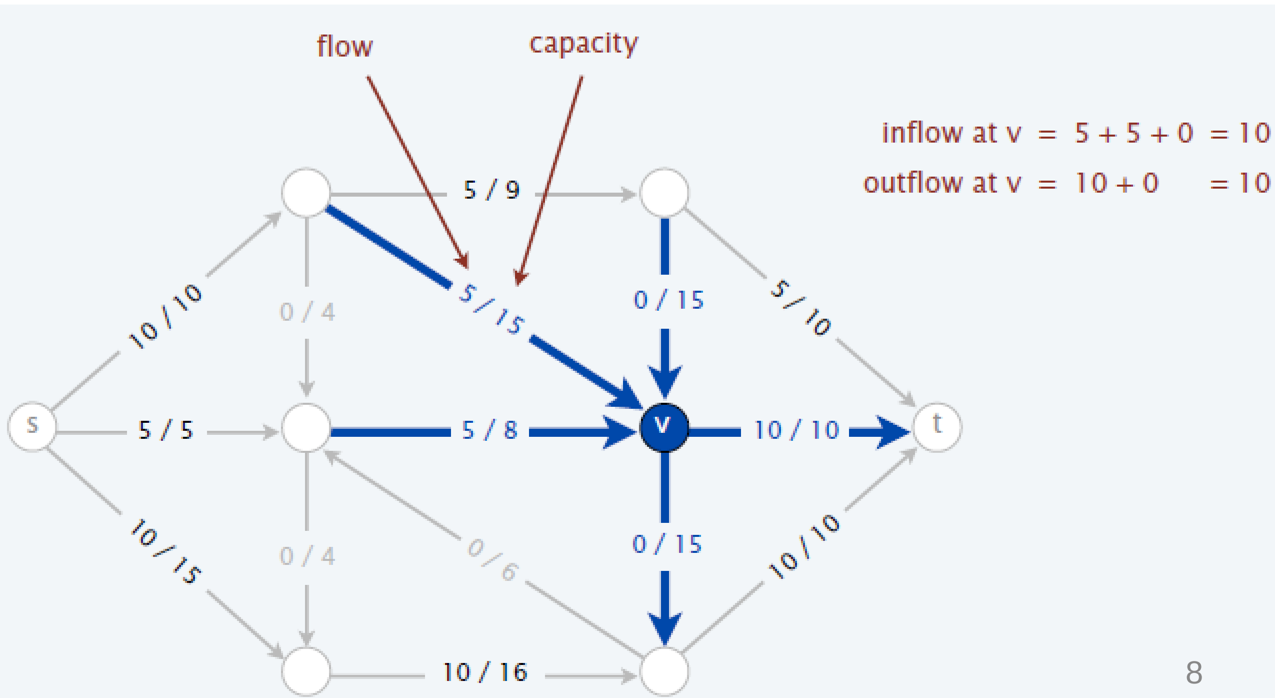
$$cap(A, B) = 45\ (20 + 25)$$



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 $v \in V - \{s,t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

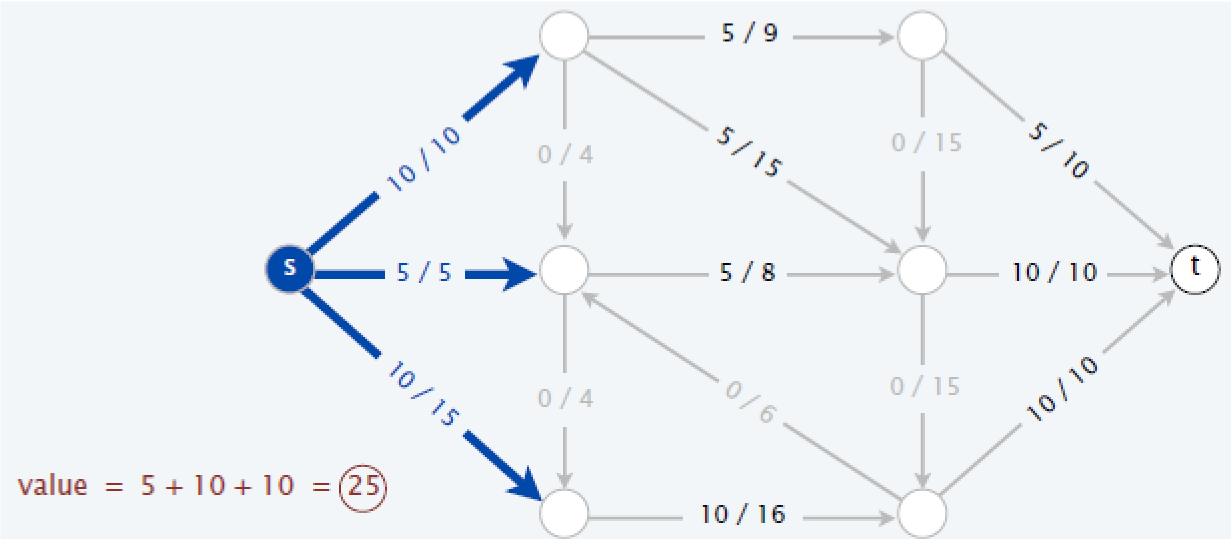


Maximum-Flow Problem

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

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

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



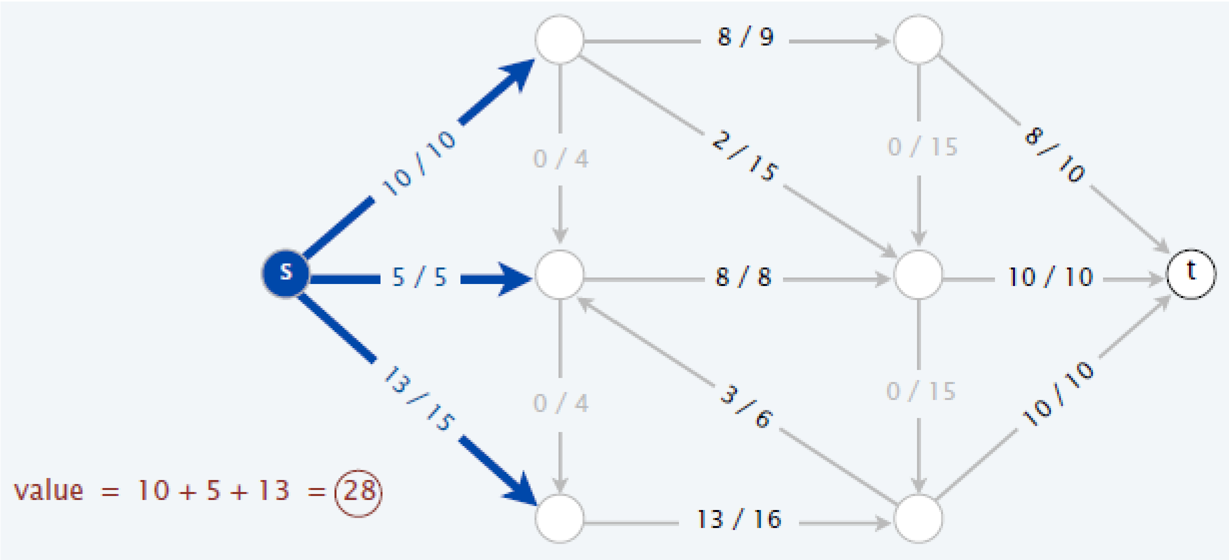
Maximum-Flow Problem

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 $v \in V - \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

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

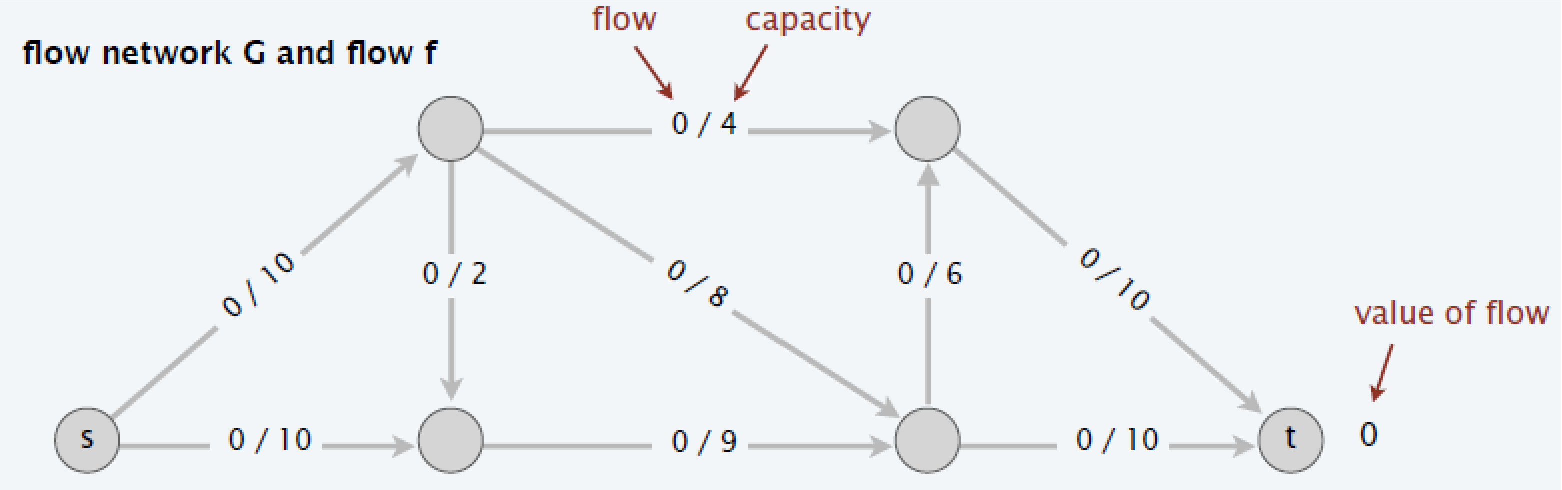
Max-flow problem. Find a flow of maximum value?



Towards a Max-Flow Algorithm

Greedy algorithm.

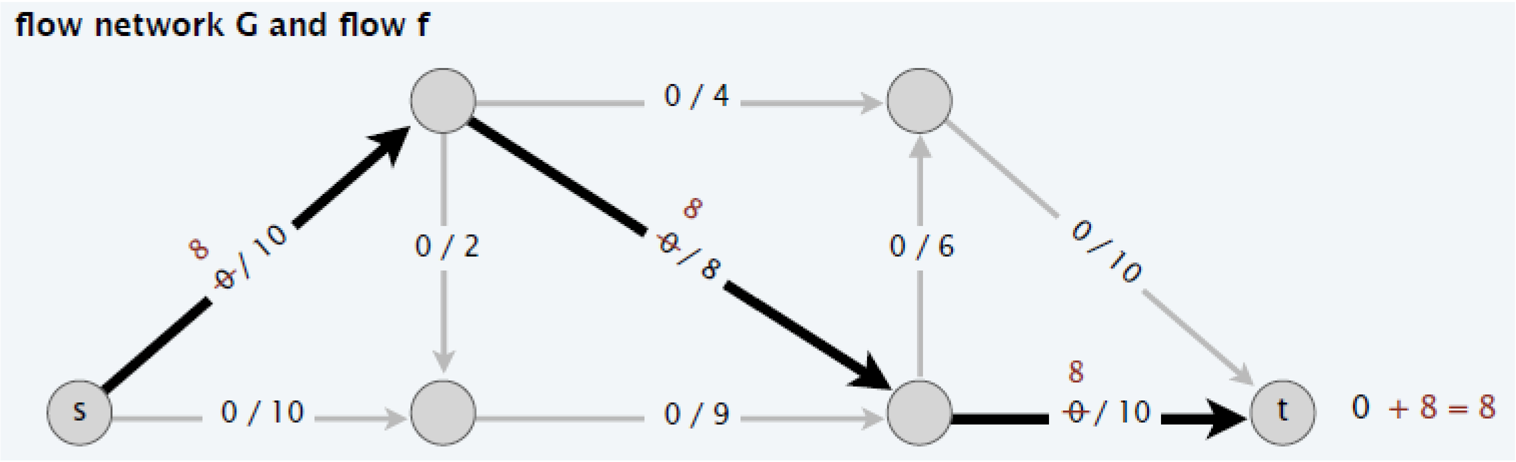
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \to t$ path P where each edge has $f(e) \leq c(e)$.
- Augment flow along path P.
- Repeat until get stuck.



Towards a Max-Flow Algorithm

Greedy algorithm.

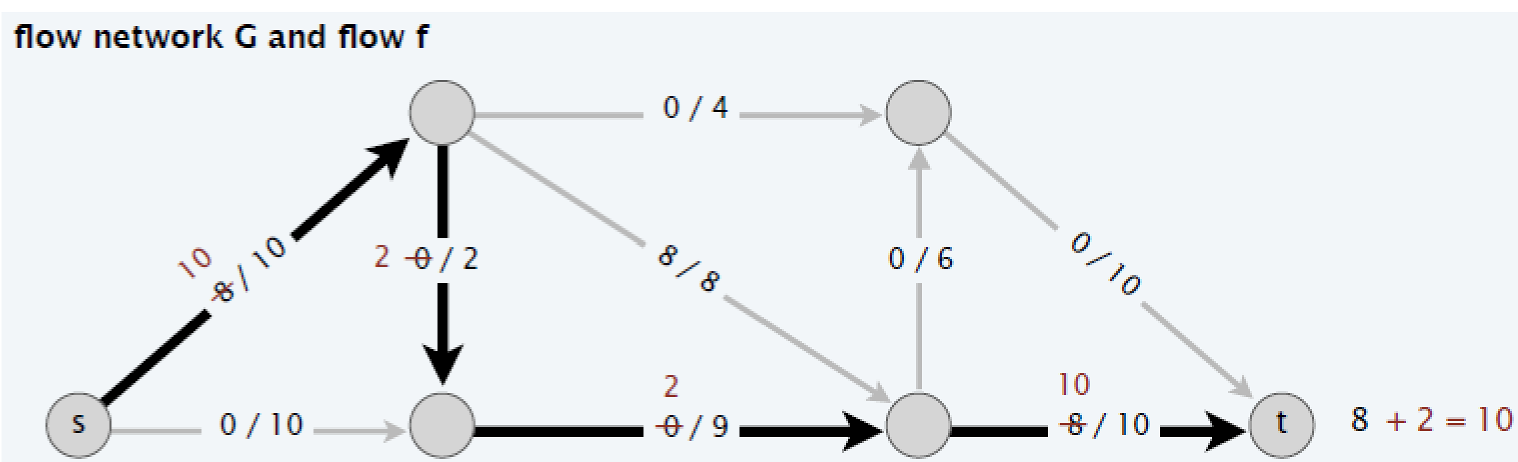
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \to t$ path P where each edge has $f(e) \leq c(e)$.
- Augment flow along path P.
- Repeat until get stuck.



Towards a Max-Flow Algorithm

Greedy algorithm.

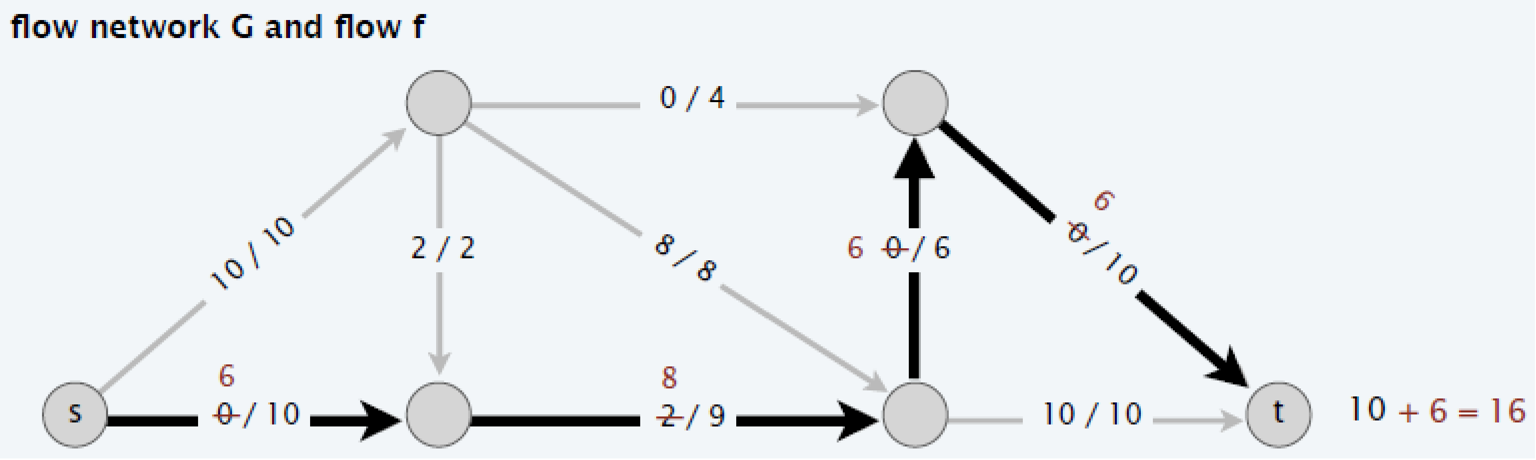
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) \leq c(e)$.
- Augment flow along path P.
- Repeat until get stuck.



Towards a Max-Flow Algorithm

Greedy algorithm.

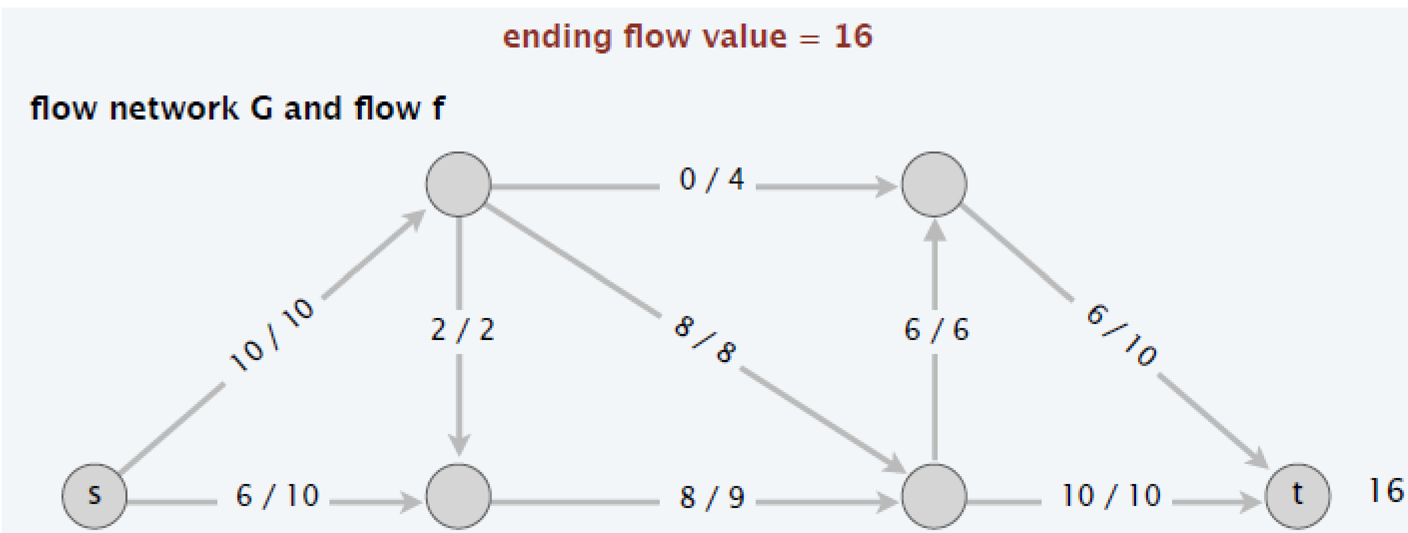
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \to t$ path P where each edge has $f(e) \leq c(e)$.
- Augment flow along path P.
- Repeat until get stuck.



Towards a Max-Flow Algorithm

Greedy algorithm.

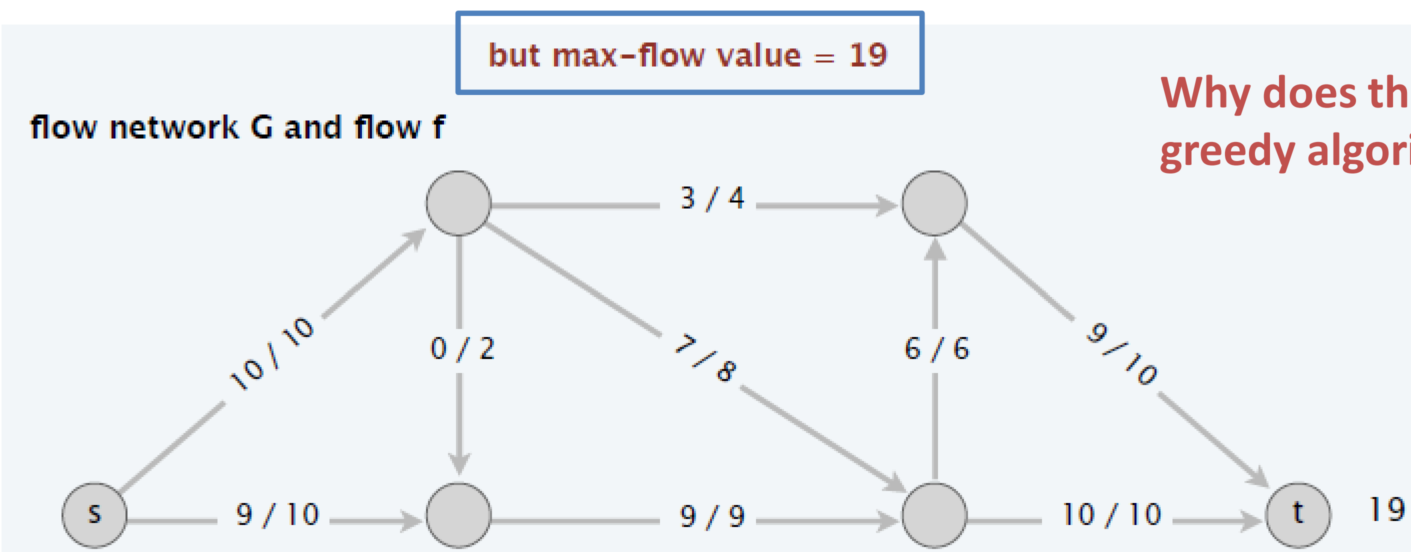
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \to t$ path P where each edge has $f(e) \leq c(e)$.
- Augment flow along path P.
- Repeat until get stuck.



Towards a Max-Flow Algorithm

Greedy algorithm.

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



Why does the greedy algorithm fail?

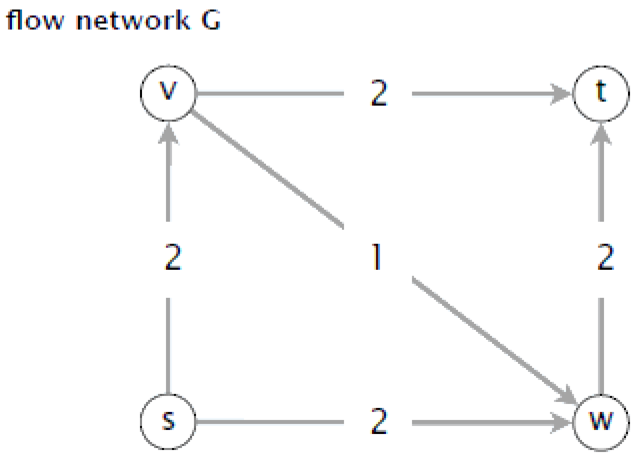
Why the Greedy Algorithm Fails

Q. Why does the greedy algorithm fail?

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

Ex.

- The max flow is unique; flow on edge (v, w) is zero.
- Greedy algorithm could choose $s \to v \to w \to t$ for first augmenting path.

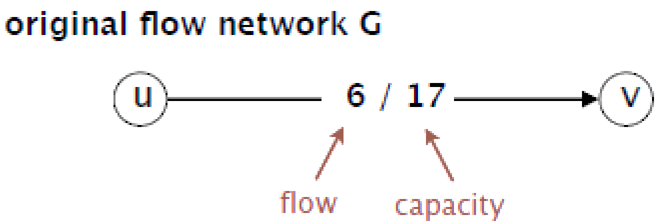


Need some mechanism to “undo” bad decision.

Residual Network

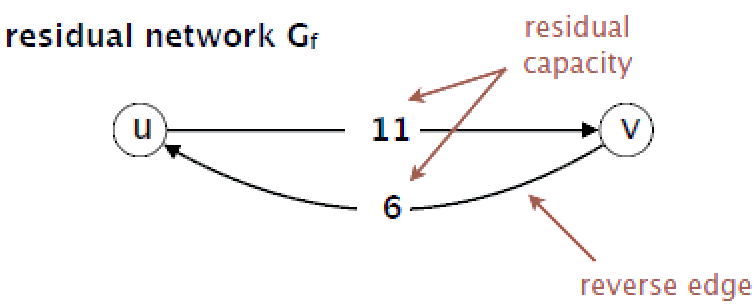
Original edge. $e = (u, v) \in E$.

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



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

- “Undo” flow sent.



Residual capacity.

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

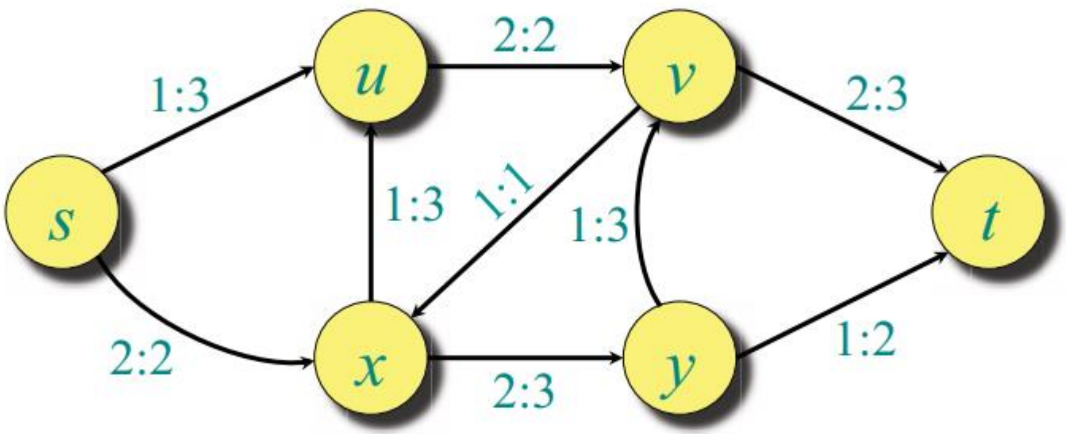
Residual network. $G_f = (V, E_f, s, t, c_f)$.

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

Edges with positive residual capacity

Flow and Residual Network

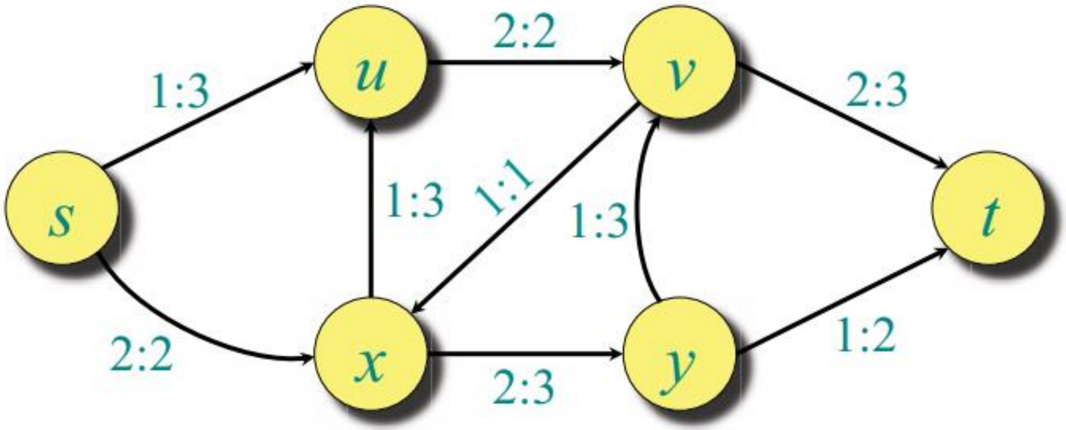
Let f be a flow on $G = (V, E)$:



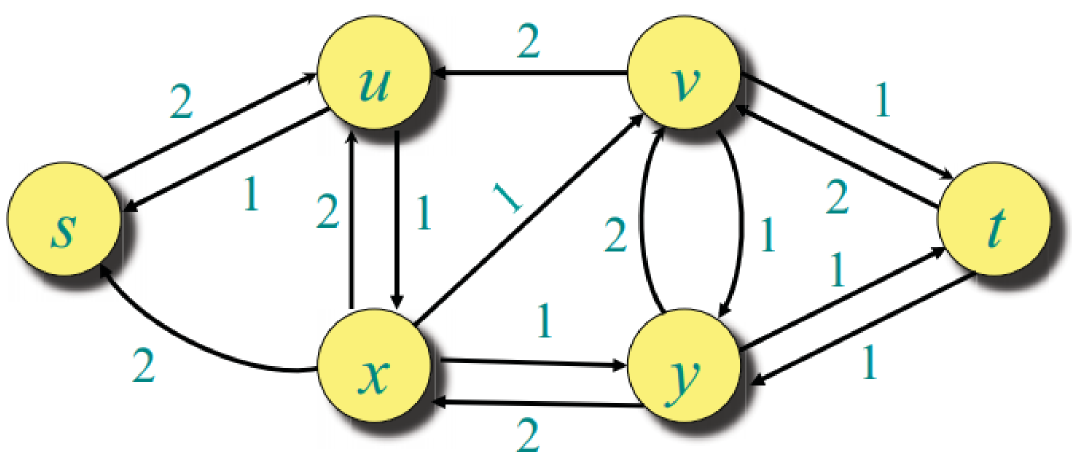
The residual network $G_f(V, E_f)$:

Flow and Residual Network

Let f be a flow on $G = (V, E)$:



The residual network $G_f(V, E_f)$:



Augmenting Path

Def. An augmenting path is a simple $s \to t$ 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 call Augment, the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

Augmenting Path

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

Augment (f, c, P)

$b \leftarrow$ bottleneck capacity of path P.
For each edge $e \in P$
  If $(e \in E)$
    $f[e] \leftarrow f[e] + b$.
  Else
    $f[e^{reverse}] \leftarrow f[e^{reverse}] - b$.
Return f.

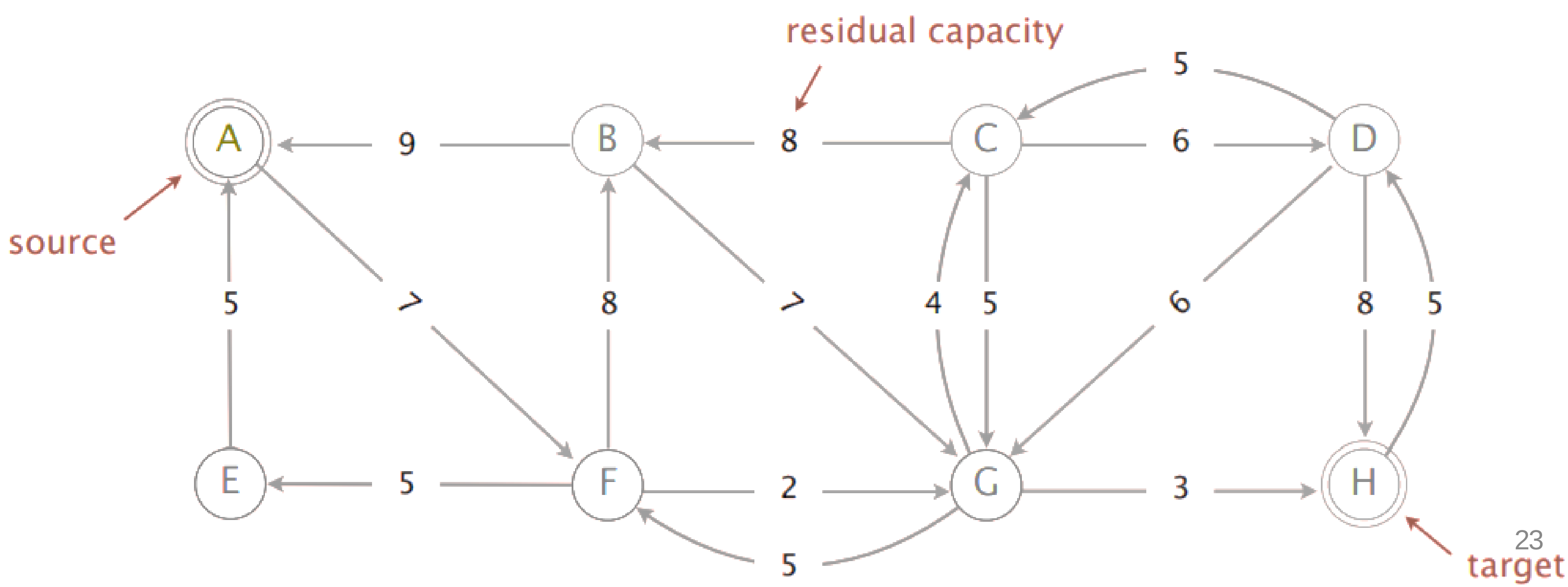
Augmenting Path

Which is the augmenting path of highest bottleneck capacity?

A. $A \to F \to G \to H$

B. $A \to F \to B \to G \to H$

C. $A \to F \to B \to G \to C \to D \to H$



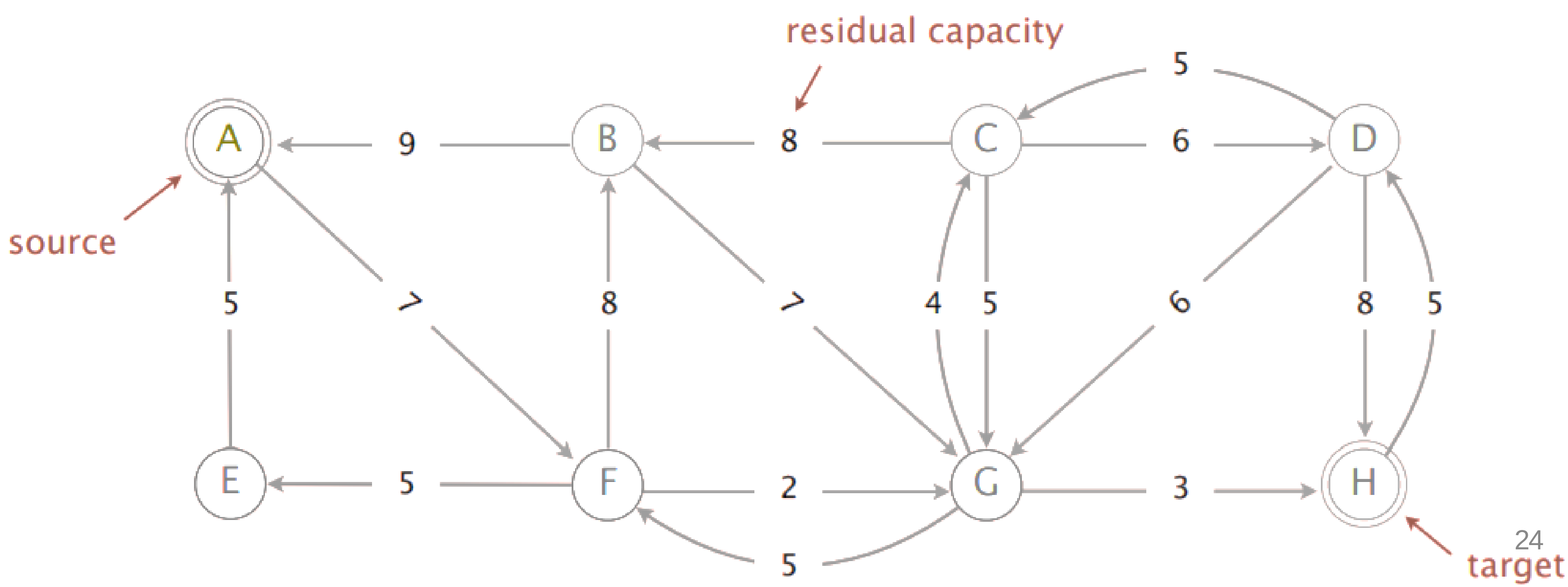
Augmenting Path

Which is the augmenting path of highest bottleneck capacity?

A. A → F → G → H (2)

B. A → F → B → G → H (3)

C. A → F → B → G → C → D → H (4)



Ford-Fulkerson Algorithm

Ford-Fulkerson augmenting path algorithm.

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

Ford-Fulkerson (G)

For each edge $e \in E$: $f[e] \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to f.

While (there exists an $s \to t$ augmenting path P in G_f

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

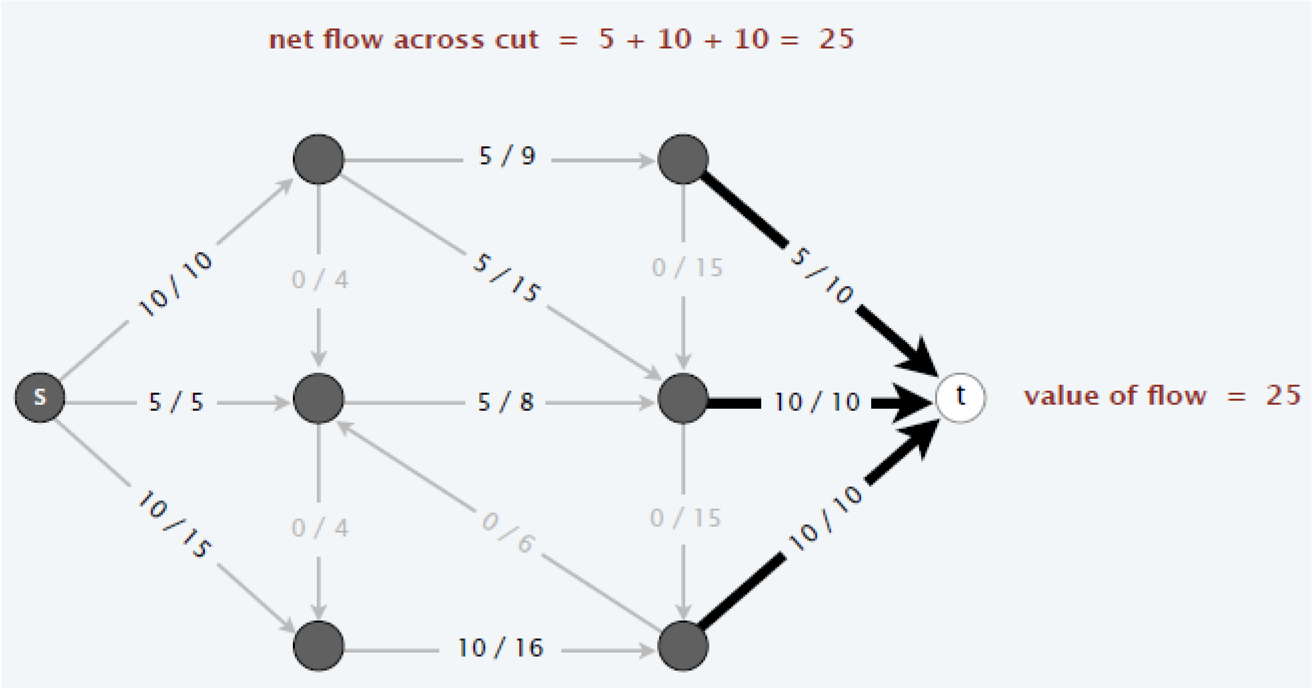
  Update G_f.

Return f.

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 cut (A, B).

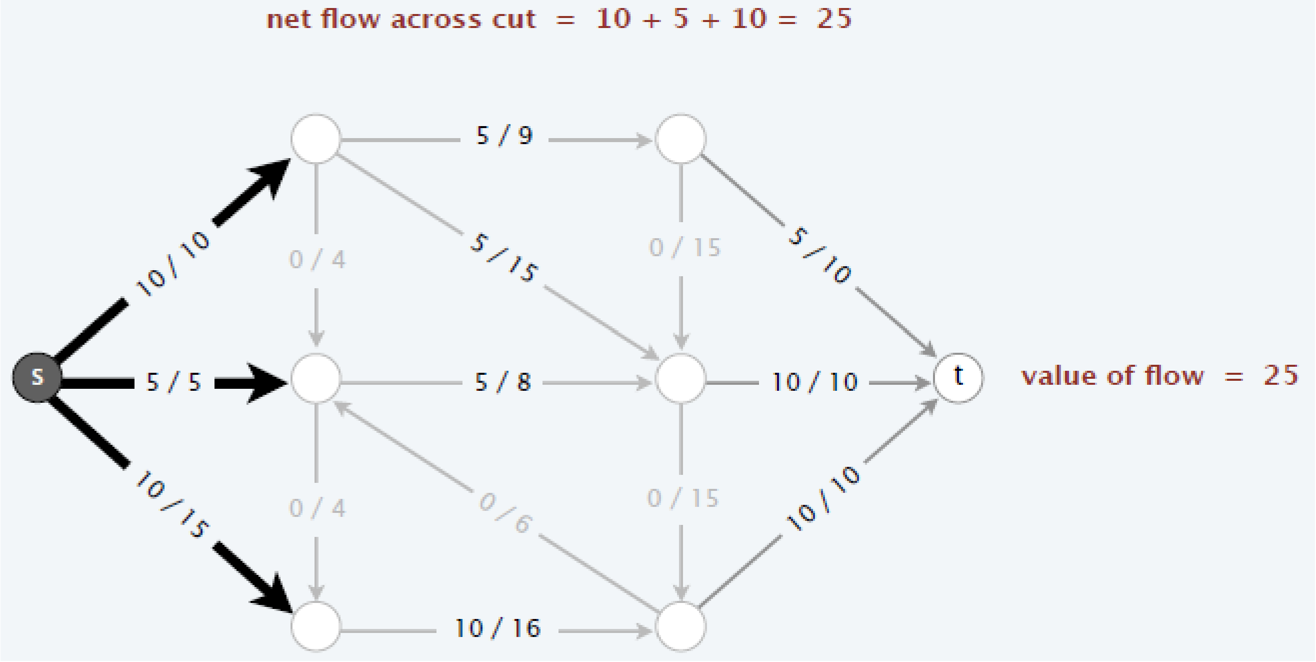
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



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 cut (A, B).

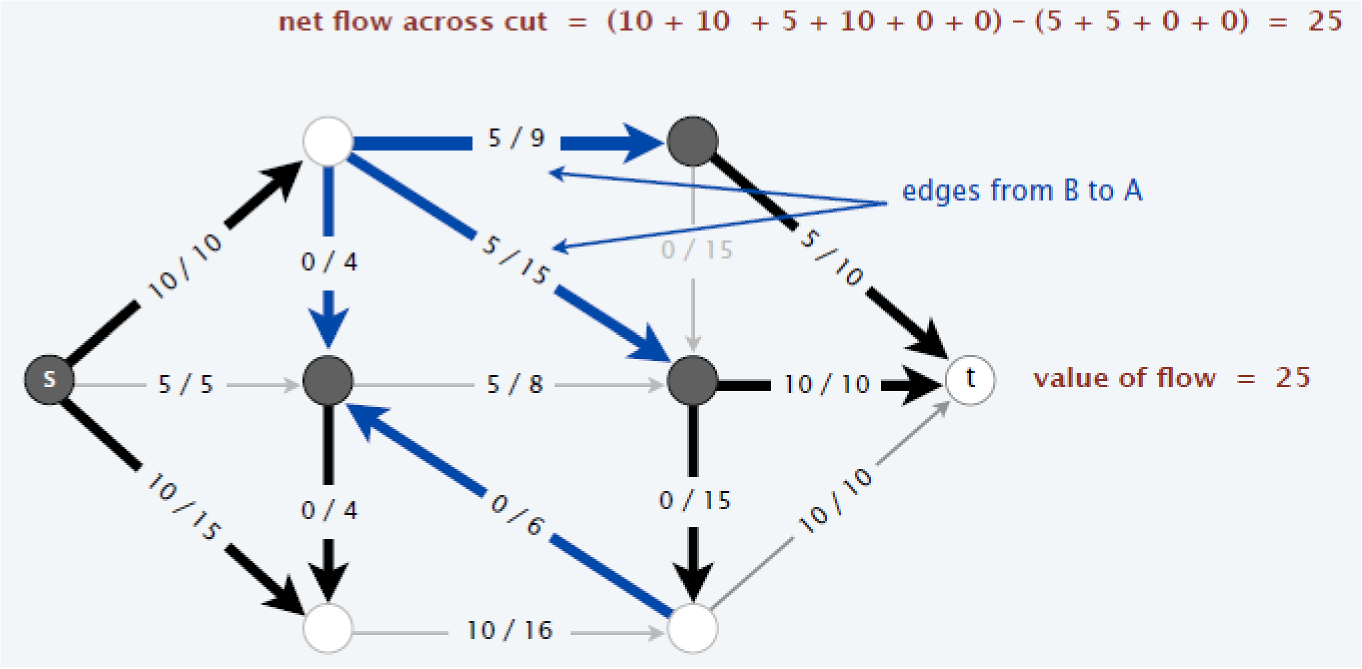
$$val(f) = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e)$$



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 cut (A, B).

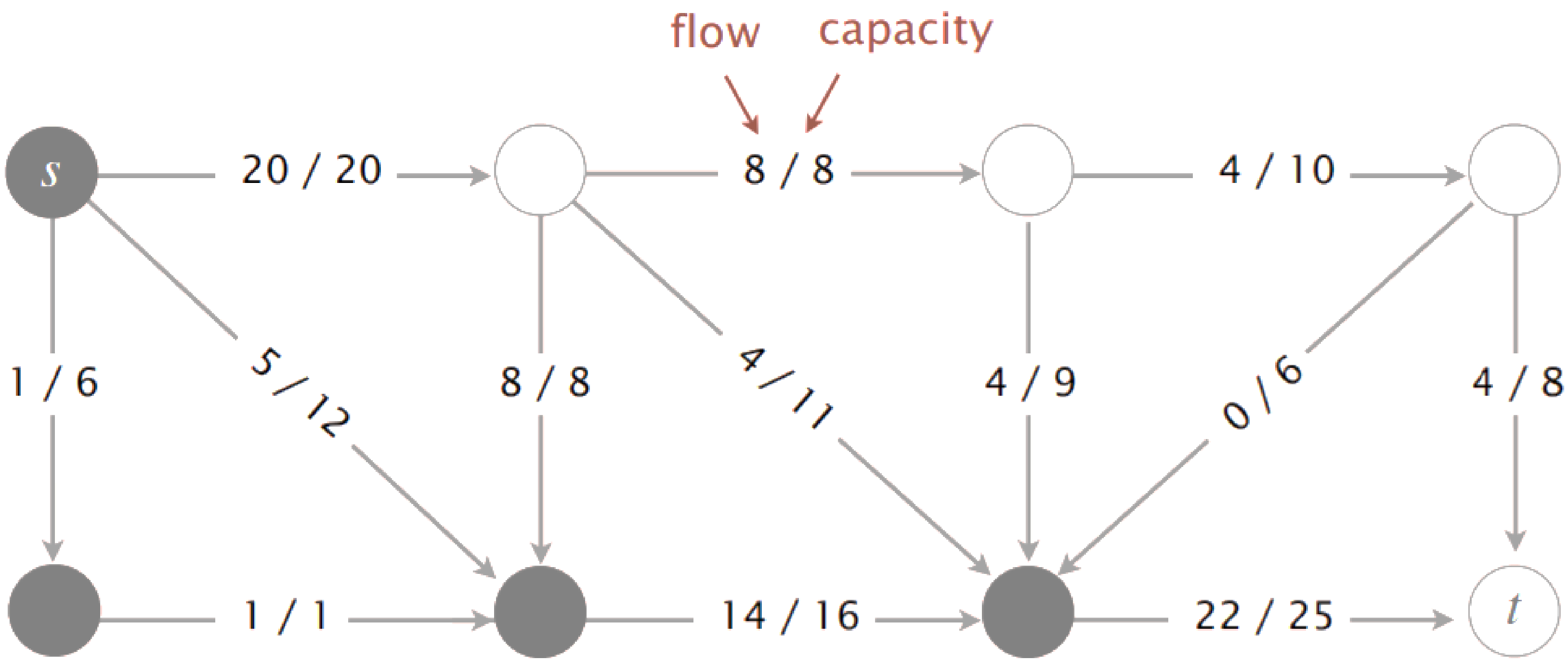
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Relationship Between Flows and Cuts

What is the net flow across the given *st*-cut?

$$val(f) = 26\ (20 + 22 - 8 - 4 - 4)$$



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 cut (A, B).

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

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

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

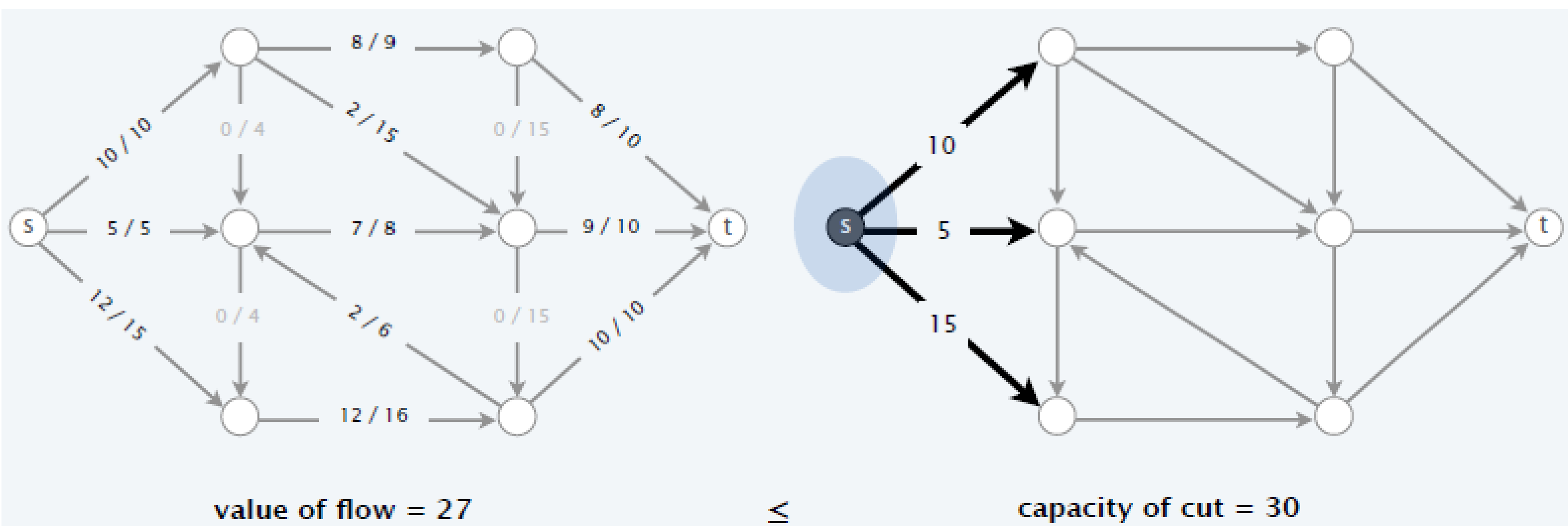
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

By flow conservation, all terms except for *s* are 0

Relationship Between Flows and Cuts

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

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

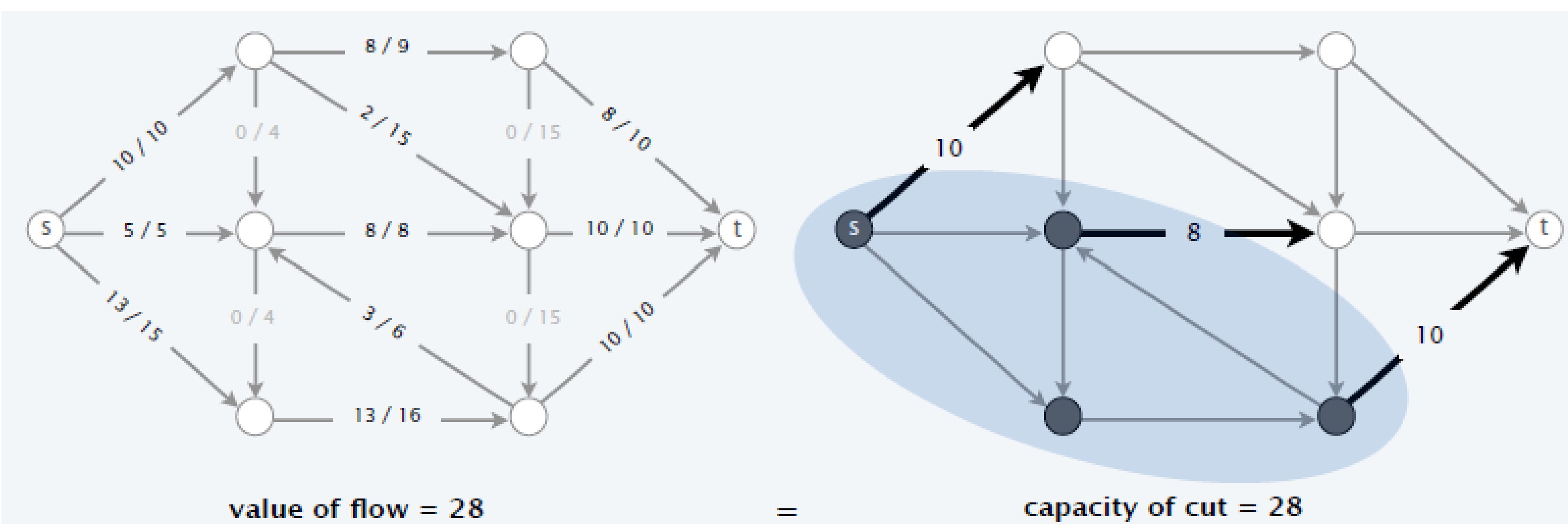


Certificate of Optimality

Corollary. Let f be a flow and let (A, B) be any cut. If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

- For any flow f': $val(f') \le cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \ge val(f) = cap(A, B)$.



Computing a Minimum Cut from a Maximum Flow

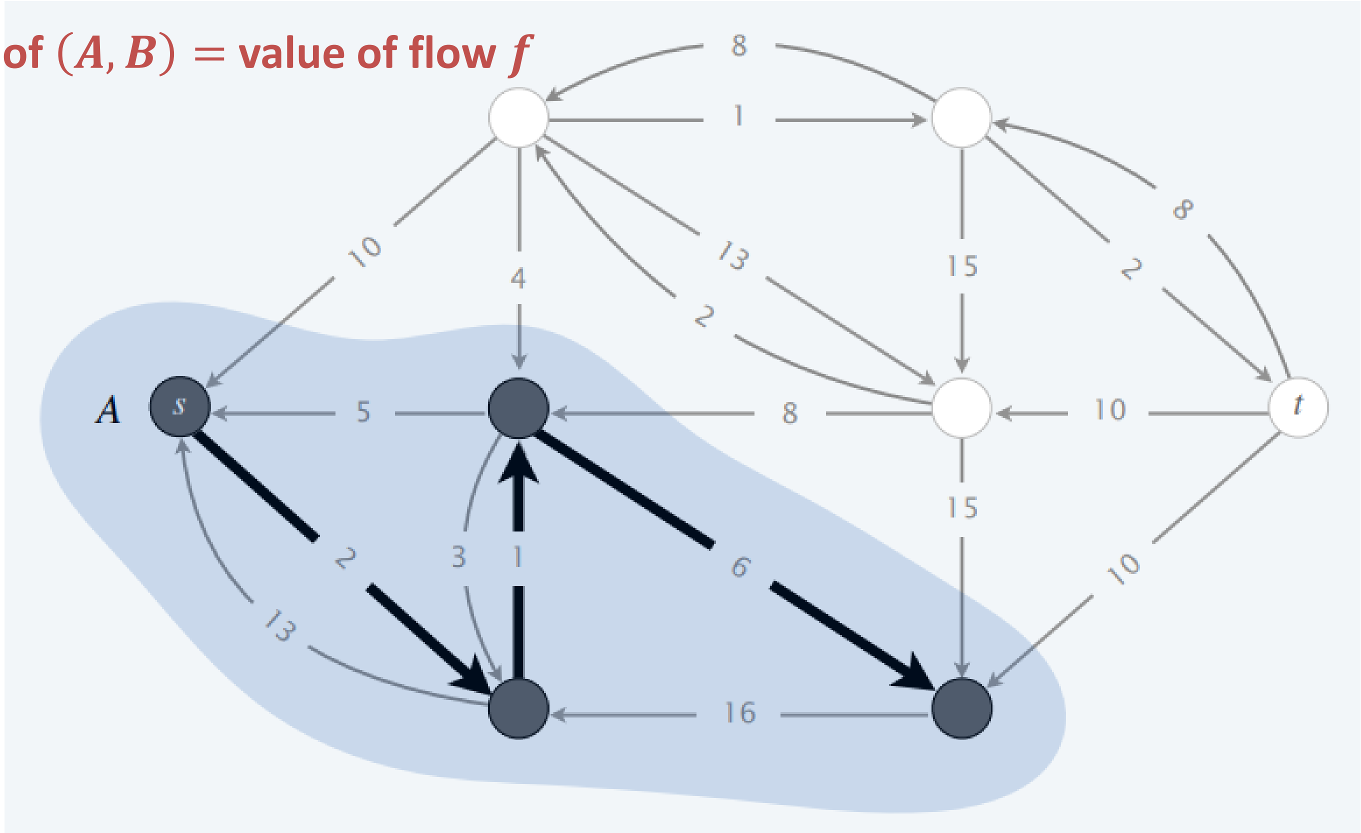
Theorem. Given any max flow f, can compute a min cut (A, B) in $O(m)$ time.

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.

Pf. Let $A =$ set of nodes reachable from s in residual network G_f.

Capacity of $(A, B) =$ value of flow f



Max-Flow Min-Cut Theorem

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

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Pf. The following three conditions are equivalent for any flow f:

I. There exists a cut (A, B) such that $cap(A, B) = val(f)$.

II. f is a max flow.

III. There is no augmenting path with respect to f.

[I $\Longrightarrow$ II]

- Suppose that (A, B) is a cut such that $cap(A, B) = val(f)$.
- Then, for any flowf': $val(f') \leq cap(A, B) = val(f)$.
- Thus, f is a max flow.

Max-Flow Min-Cut Theorem

Augmenting path theorem. A flow f is a max flow iff no augmenting paths.
Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Pf. The following three conditions are equivalent for any flow f:

I. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
II. f is a max flow.
III. There is no augmenting path with respect to f.

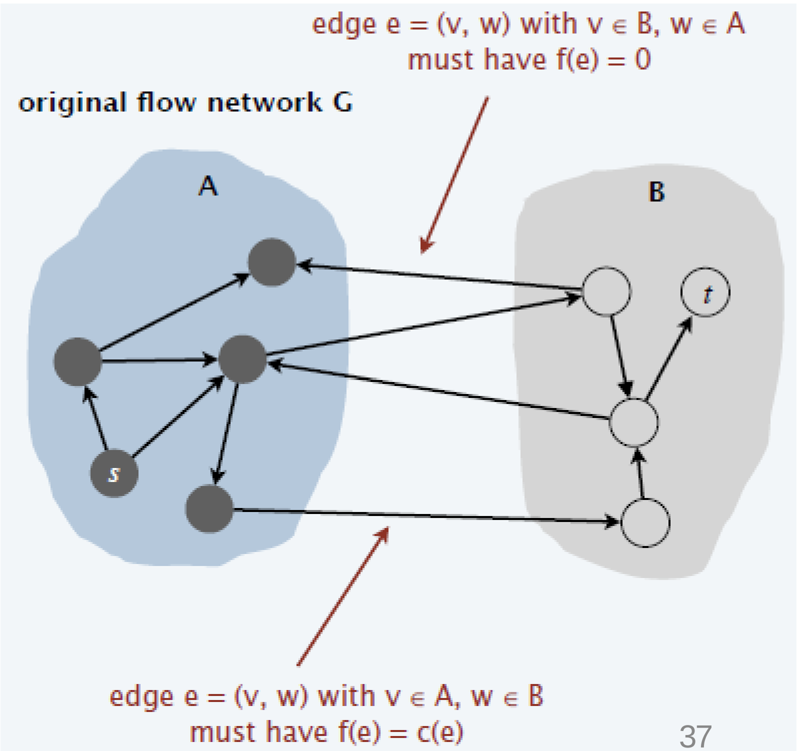
[II ⟹ III] We prove contrapositive: ~III ⟹ ~II.

- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a max flow.

Max-Flow Min-Cut Theorem

[III $\Longrightarrow$ I]

- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual network G_f.
- By definition of cut A: $s \in A$.
- By definition of flow f: $t \notin A$.



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

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

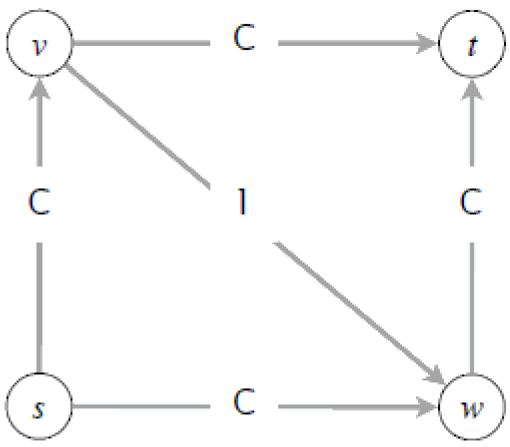
$$= cap(A, B)$$

Bad Case for Ford-Fulkerson

If max capacity is C, then algorithm can take $\geq C$ iterations.

- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- …
- $s \to v \to w \to t$
- $s \to w \to v \to t$

each augmenting path
sends only 1 unit of flow
(# augmenting paths = 2C)



Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

Pathology. If capacities are irrational, algorithm does not guarantee to terminate (or converge to correct answer)!

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

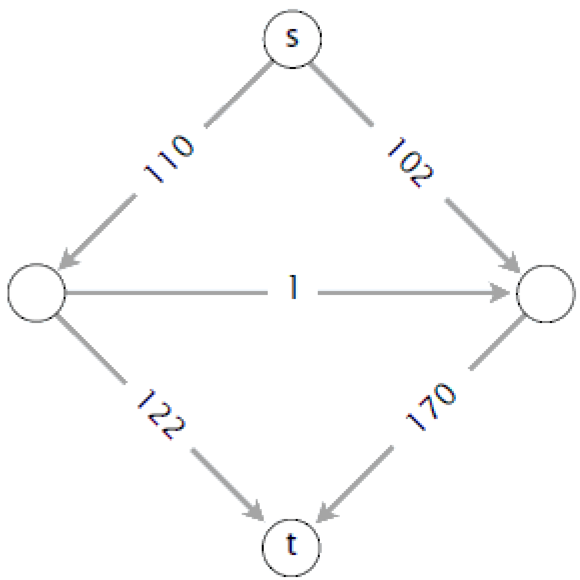
Choose augmenting paths with:

- Max bottleneck capacity ("fattest").
- Sufficiency large bottleneck capacity.
- Fewest edges.

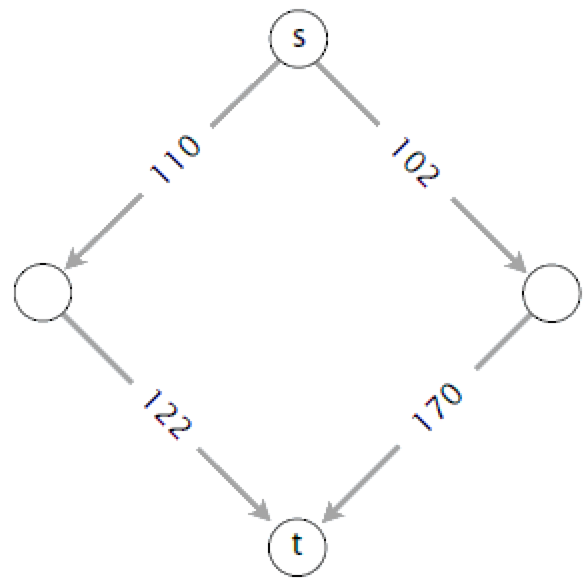
Capacity-Scaling Algorithm

Intuition. Choose augmenting path with highest bottleneck capacity: it increases flow by max possible amount in given iteration.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the part of the residual network consisting of only those arcs with capacity $\geq \Delta$.



G_f



$G_f(\Delta)$, $\Delta = 100$

Capacity-Scaling Algorithm

Capacity-Scaling (G)

For each 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 $s \rightarrow t$ path P in $G_f(\Delta)$
    $f \leftarrow$ Augment(f, c, P).
    Update $G_f(\Delta)$.
  $\Delta \leftarrow \Delta/2$.

Return f.

Capacity-Scaling Algorithm: Proof of Correctness

Assumption: All edge capacities are integers between 1 and C.

Integrality invariant. All flows and residual capacities are integral.

Theorem. If capacity-scaling algorithm terminates, then f is a max flow.

Pf.

- By integrality invariant, when $\Delta = 1 \Longrightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.

Shortest Augmenting Path

Q. Which augmenting path?

A. The one with the fewest edges (can find via Breadth-First-Search).

Shortest-Augmenting-Path (G)

For each edge $e \in E$: $f[e] \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

While (there exists an $s \rightarrow t$ path in G_f)

$P \leftarrow$ Breadth-First-Search (G_f).

$f \leftarrow$ Augment (f, c, P)

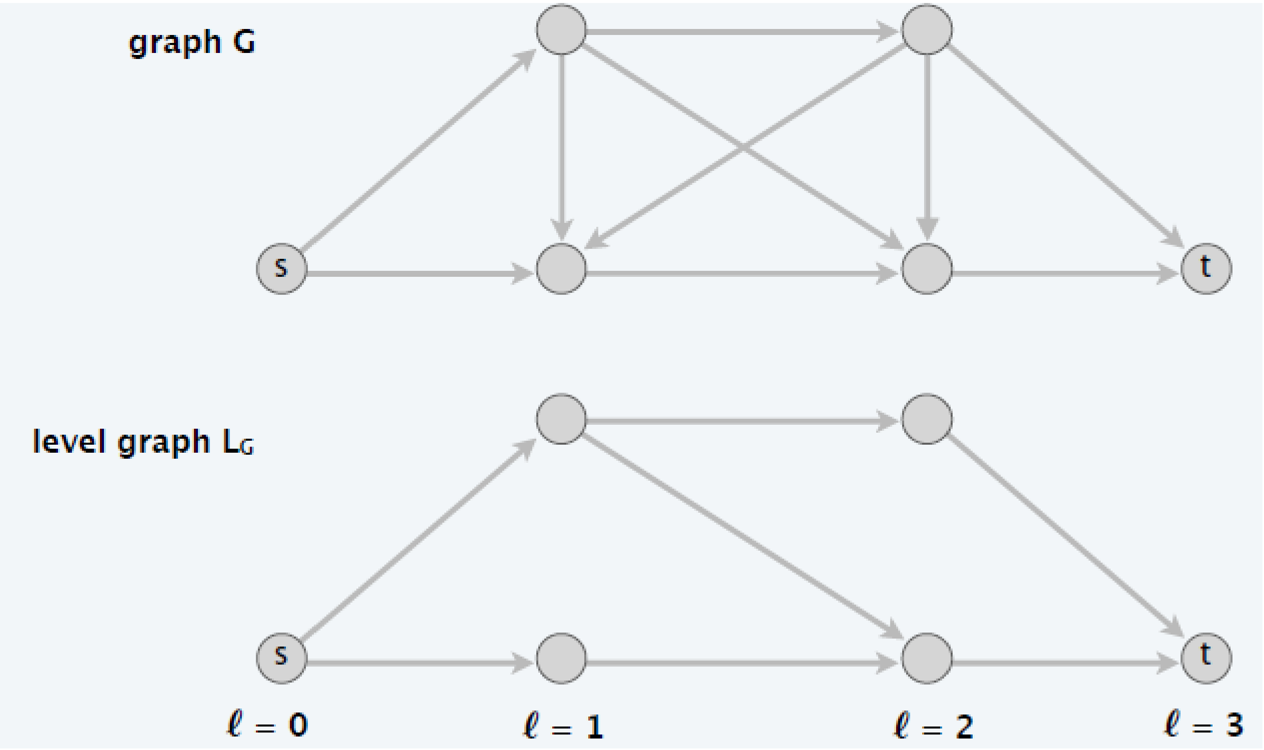
Update G_f.

Return f.

Shortest Augmenting Path: Analysis

Def. Given a digraph $G = (V, E)$ with source s, its **level graph** is defined by:

- $l(v) =$ number of edges in shortest path from s to v.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edge $(v, w) \in E$ with $l(w) = l(v) + 1$.

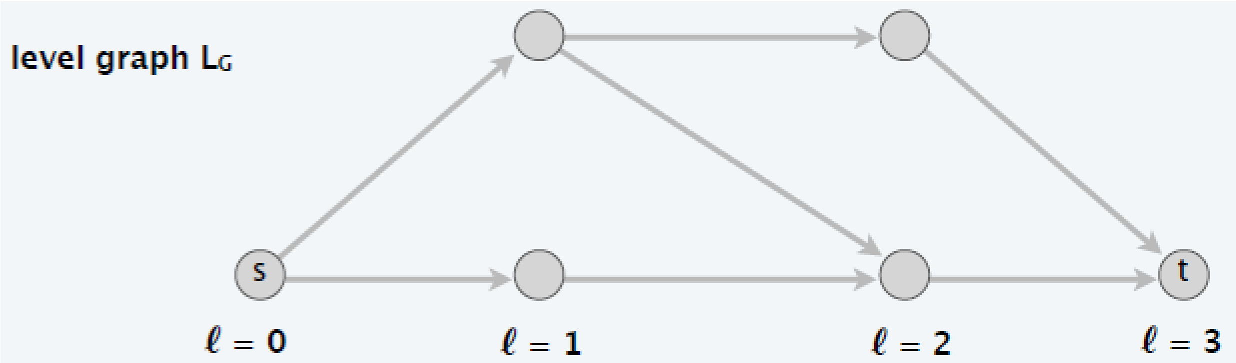


Shortest Augmenting Path: Analysis

Def. Given a digraph $G = (V, E)$ with source s, its **level graph** is defined by:

- $l(v)$ = number of edges in shortest path from s to v.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edge $(v, w) \in E$ with $l(w) = l(v) + 1$.

Key property. P is a shortest path $s \to v$ path in G iff P is an $s \to v$ path in L_G.

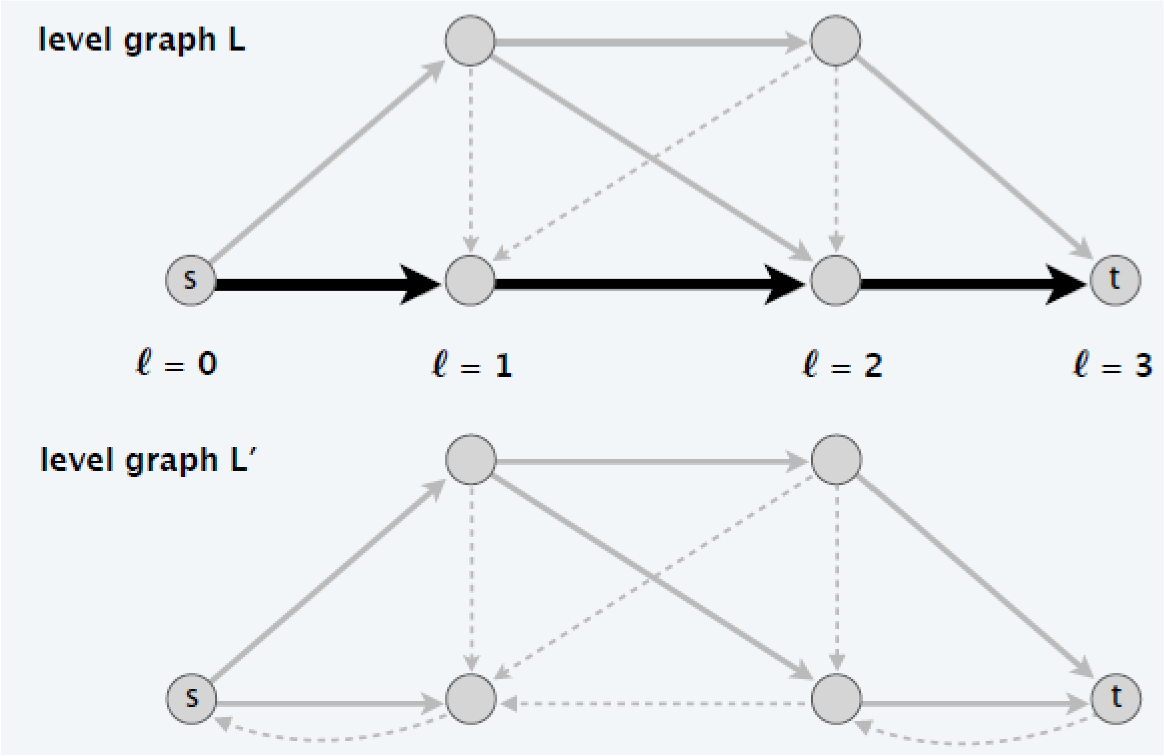


Shortest Augmenting Path: Analysis

Lemma 1. The length of a shortest augmenting path never decreases.

- Let f and f' be flow before and after a shortest-path augmentation.
- Let L and L' be level graphs of $\boldsymbol{G_f}$ and $\boldsymbol{G_{f'}}$.
- Only back edges added to $G_{f'}$.

(any path with a back edge is longer than previous length)



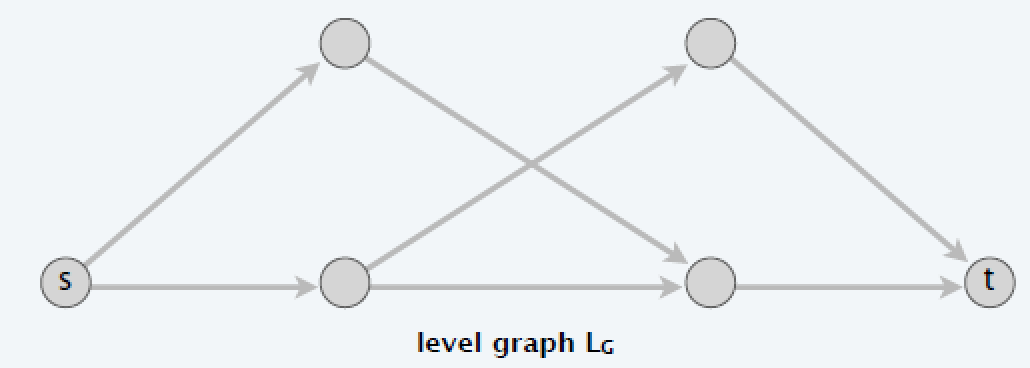
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



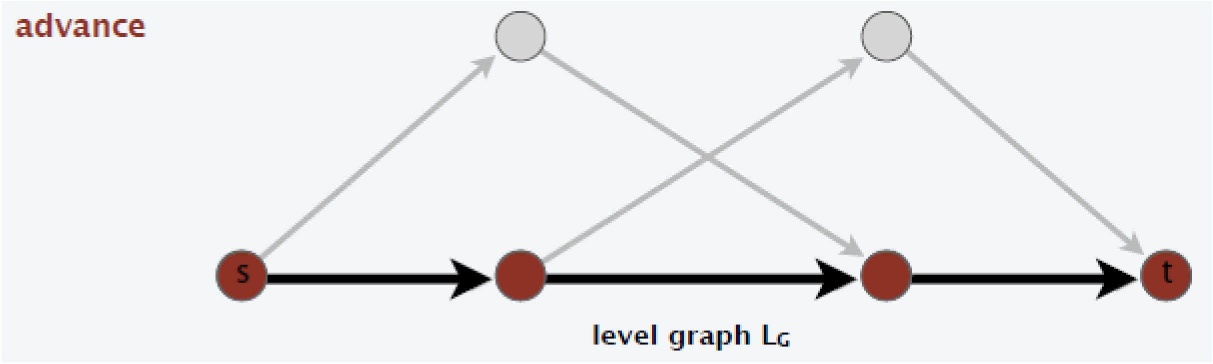
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



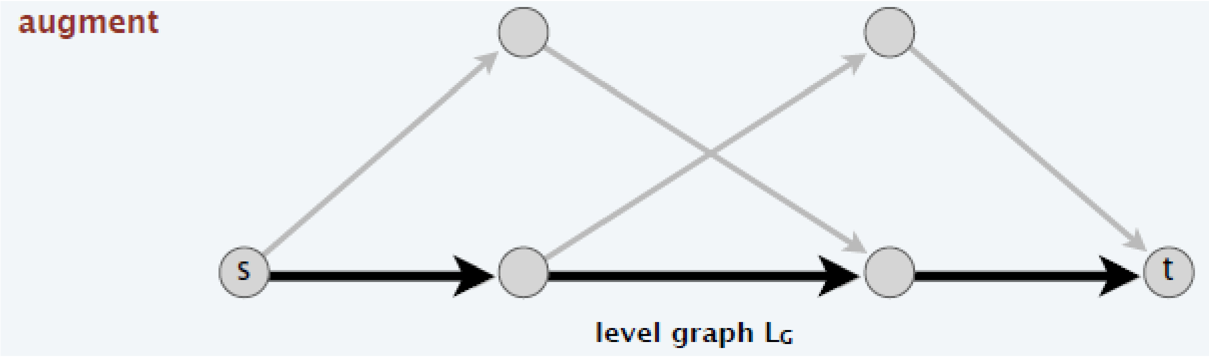
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



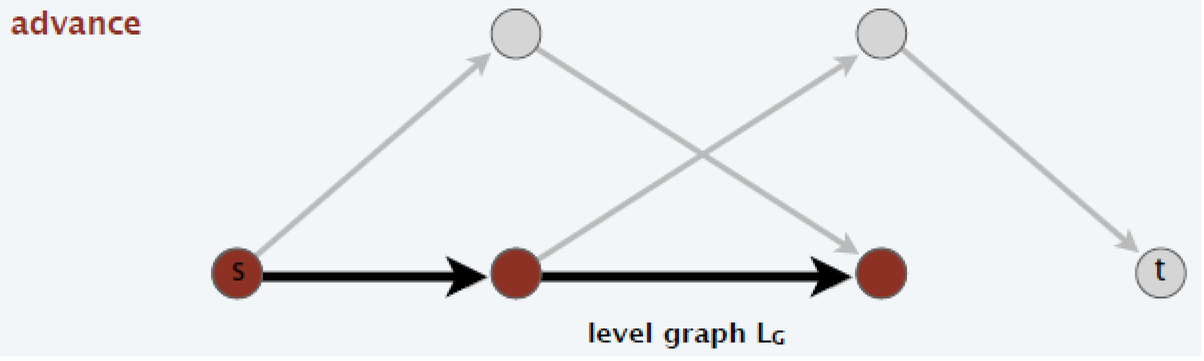
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



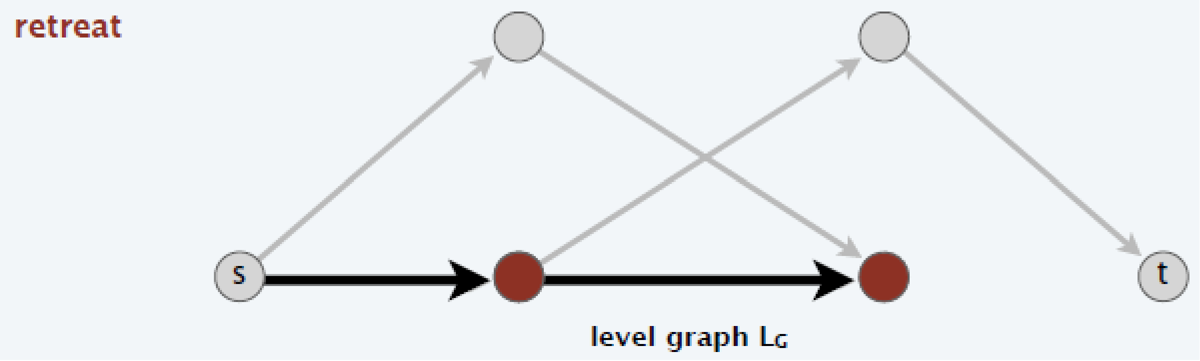
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



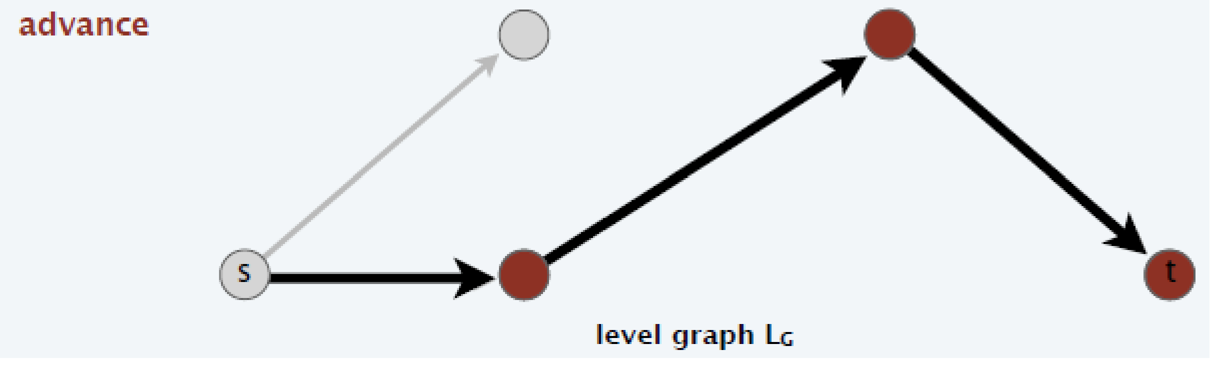
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



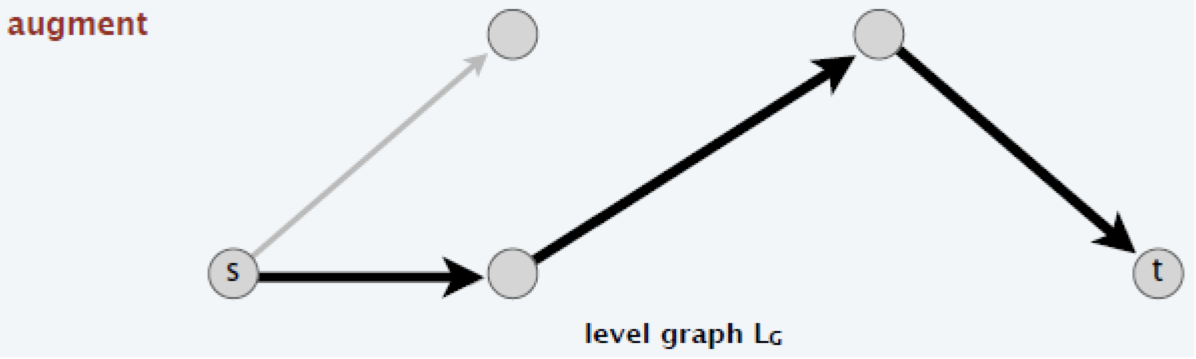
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



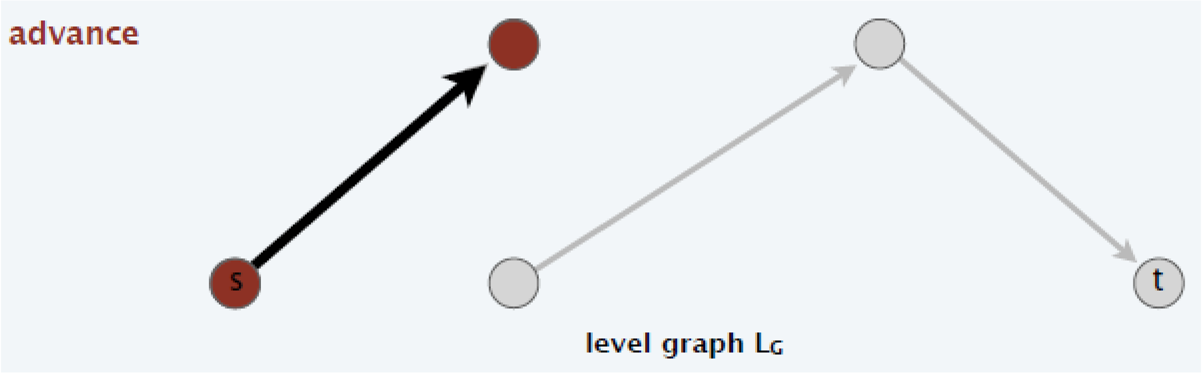
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



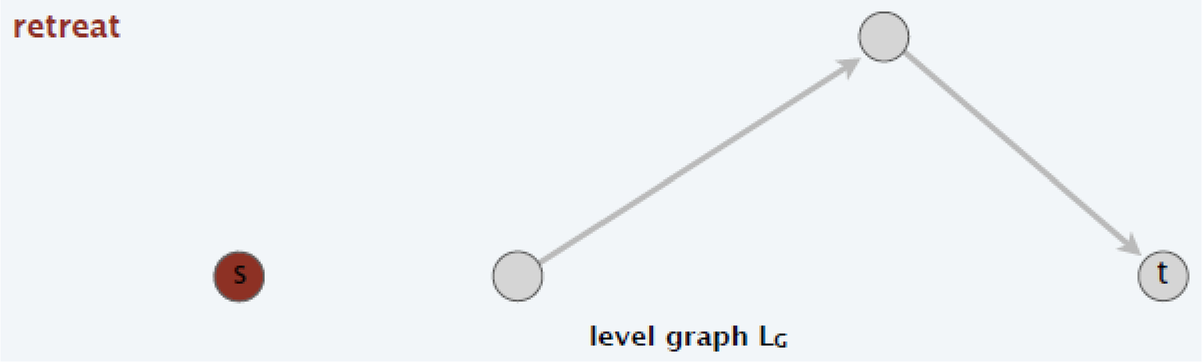
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



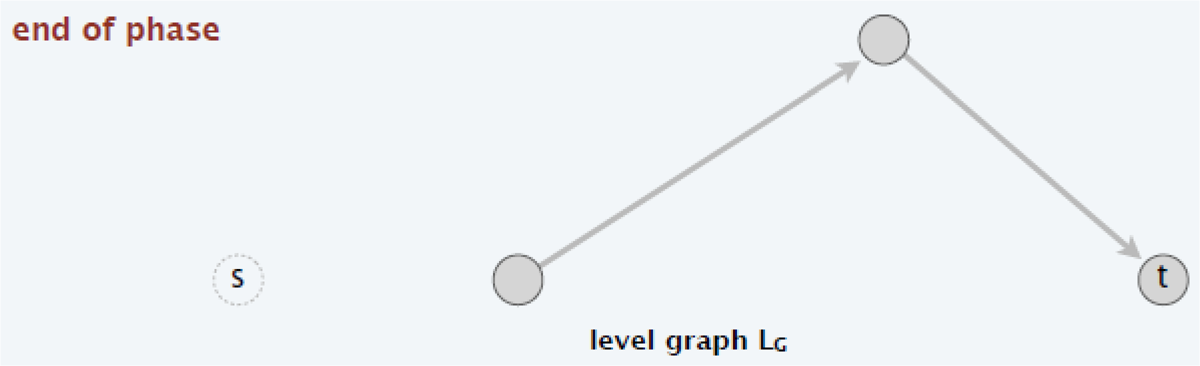
Blocking-Flow Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment and update L_G.
- If get stuck, delete node from L_G and go to previous node.



Dinitz' Algorithm

Initialize (G, f)

$L_G \leftarrow$ level-graph of G_f.
$P \leftarrow \emptyset$.
Goto Advance (s).

Retreat (v)

If $v = s$
  Stop.
Else
  Delete v (and all incident edges) from L_G.
  Remove last edge (u, v) from P .
Goto Advance (u)

Advance (v)

If $v = t$
  Augment (P).
  Remove saturated edge from L_G.
  $P \leftarrow \emptyset$.
  Goto Advance (s)

If there exists edge $(v, w) \in L_G$
  Add edge (v, w) to P.
  Goto Advance (w).
Else
  Goto Retreat (v).