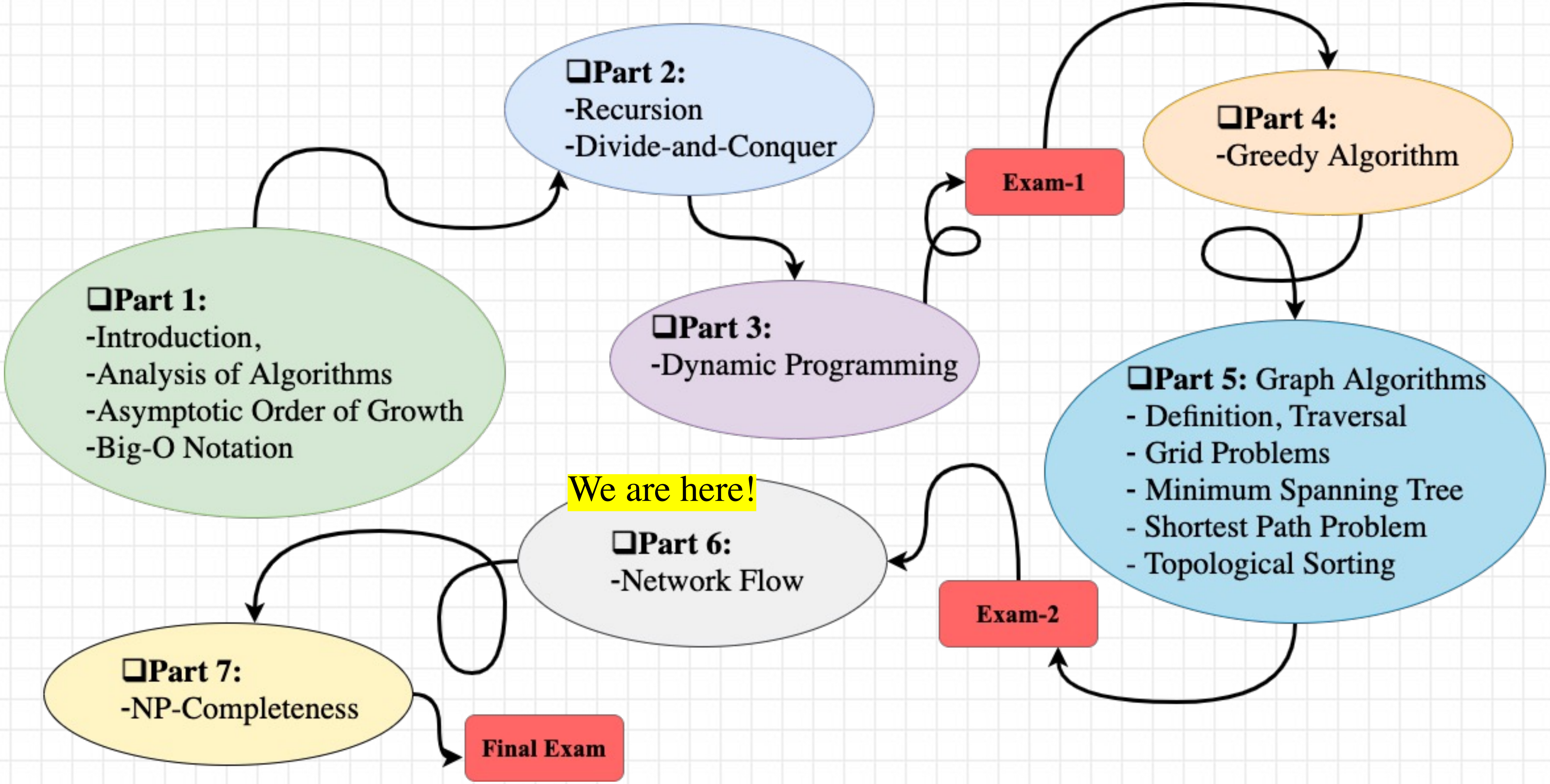
CS-3510: Design and Analysis of Algorithms

Flow Network

Instructor: Shahrokh Shahi

College of Computing
Georgia Institute of Technology
Summer 2022

Roadmap


Part 1:
-Introduction,
-Analysis of Algorithms
-Asymptotic Order of Growth
-Big-O Notation
Part 2:
-Recursion
-Divide-and-Conquer
Part 3:
-Dynamic Programming
Exam-1
Part 4:
-Greedy Algorithm
Part 5: Graph Algorithms
- Definition, Traversal
- Grid Problems
- Minimum Spanning Tree
- Shortest Path Problem
- Topological Sorting
Exam-2
We are here!
Part 6:
-Network Flow
Part 7:
-NP-Completeness
Final Exam

Graph

- Graph definition and representation
  - Adjacency matrix
  - Adjacency list
- Graph traversal
  - Breadth first search (BFS)
    - Shortest path (unweighted graphs)
    - Testing bipartiteness
    - Tree traversal (level-order)
    - Connected components
  - Depth first search (DFS)
    - Topological sorting
    - Tree traversal (in-order, pre-order, post-order)
    - Connected components

- Graph problems/algorithms
  - Minimum spanning tree (MST)
    - Kruskal (greedy)
    - Prim (greedy)
  - Shortest path (directed weighted graphs)
    - Dijkstra (greedy)
    - Bellman-Ford (dynamic programming)
    - Floyd-Warshall (dynamic programming)
  - Flow network
    - Max-flow min-cut theorem
    - Ford-Fulkerson algorithm

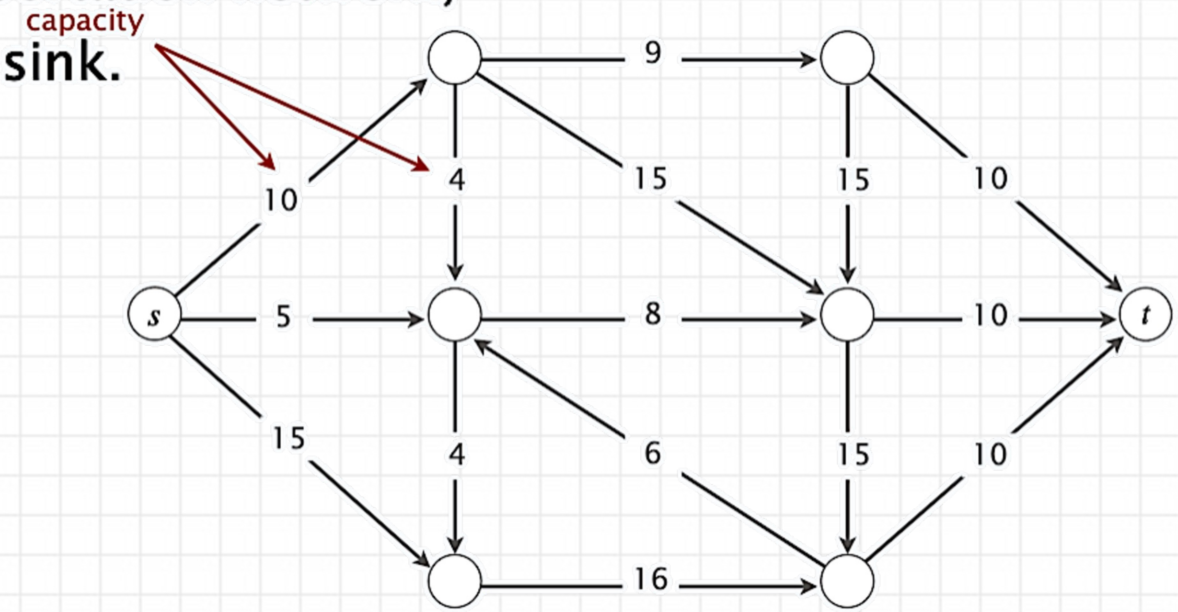
Flow Network

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

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

assume all nodes are reachable from s

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

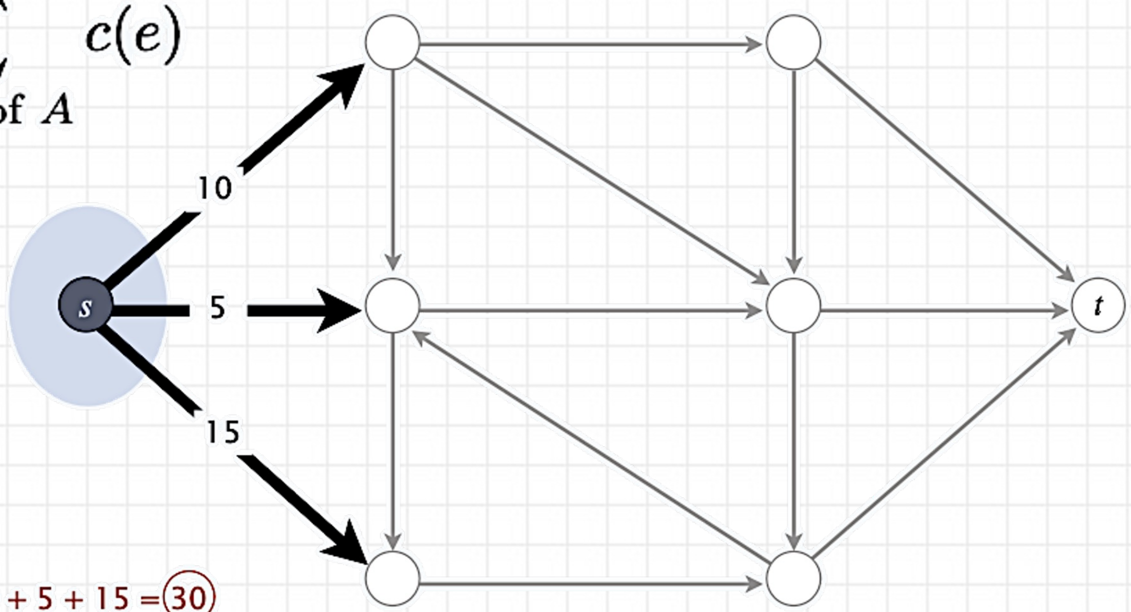


Flow Network: Min-Cut Problem

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.

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



capacity = 10 + 5 + 15 = 30

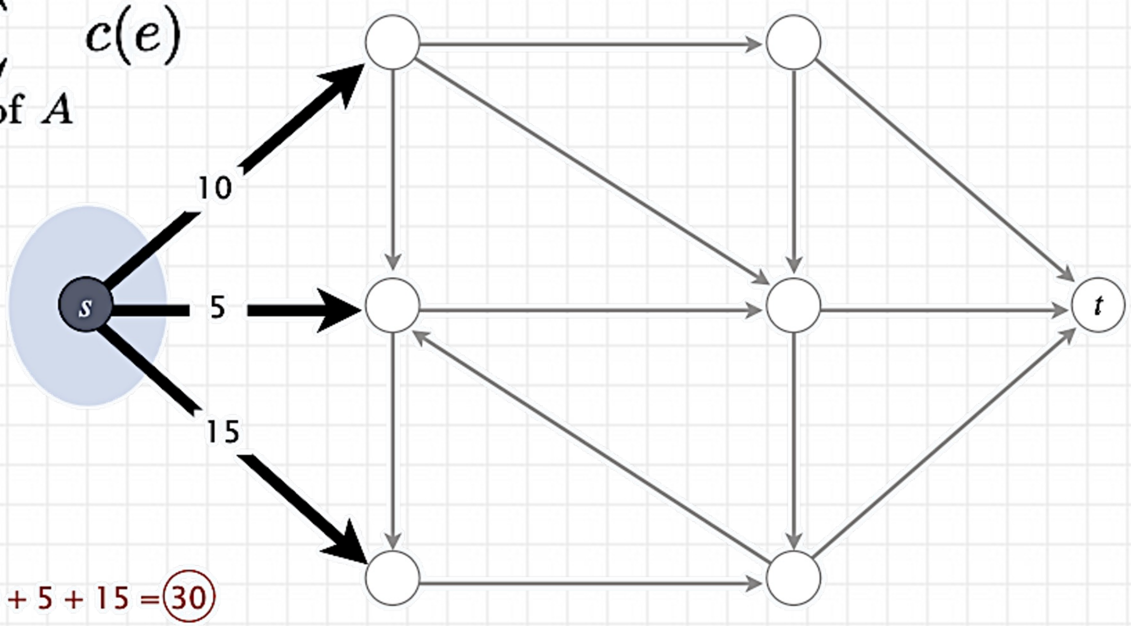
Flow Network: Min-Cut Problem

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.

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

Cut notations: (A, B) ≡ (A, V-A) ≡ (A, V\A)



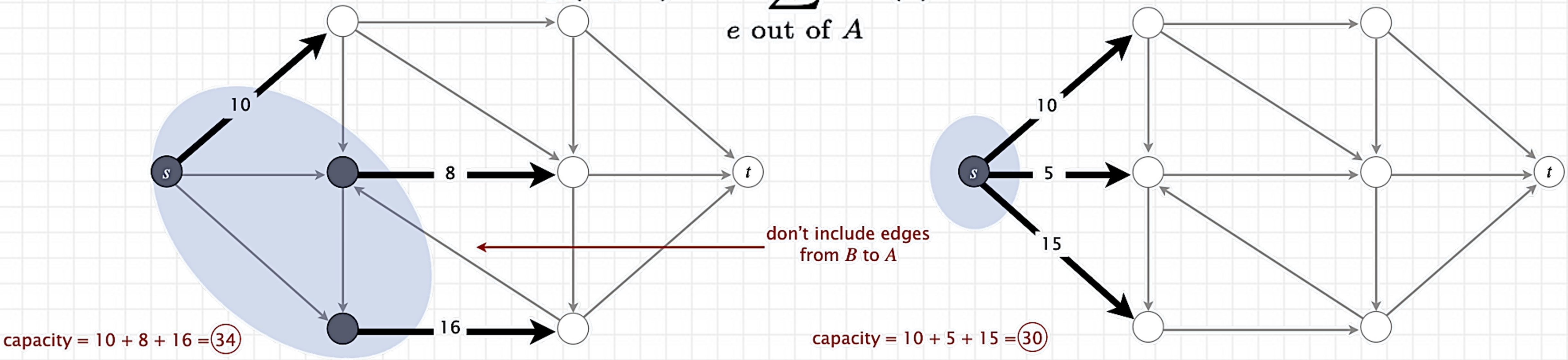
capacity = 10 + 5 + 15 = 30

Flow Network: Min-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.

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



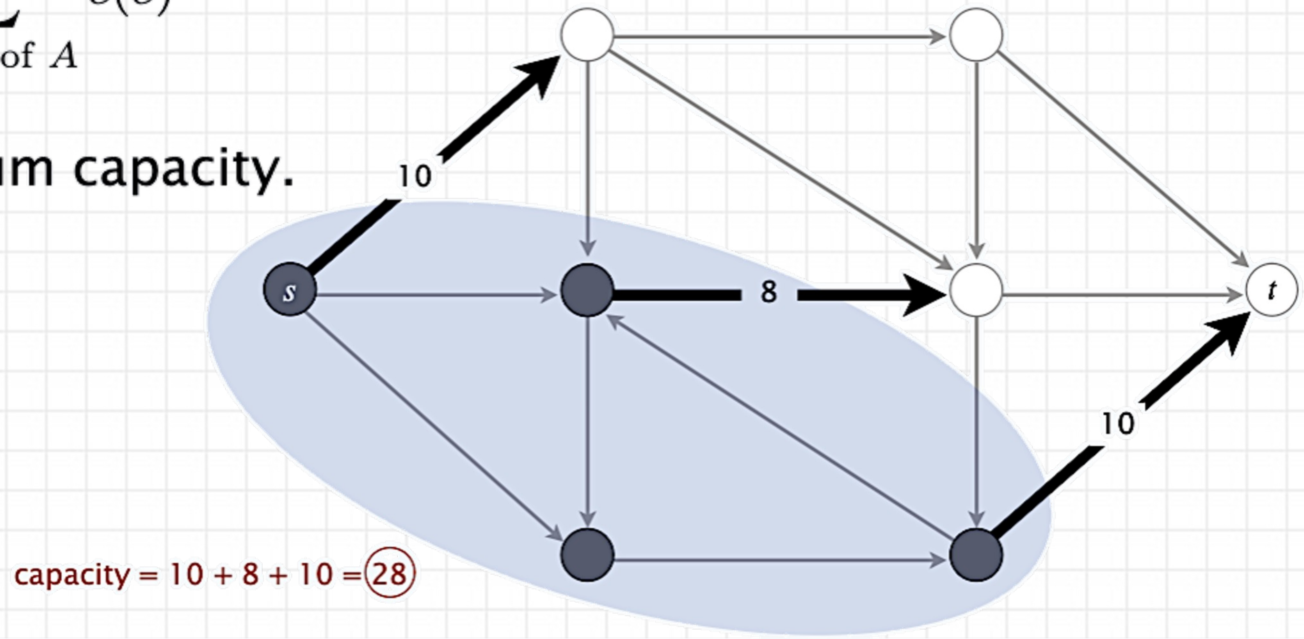
Flow Network: Min-Cut Problem

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.



Flow Network: Max-Flow Problem

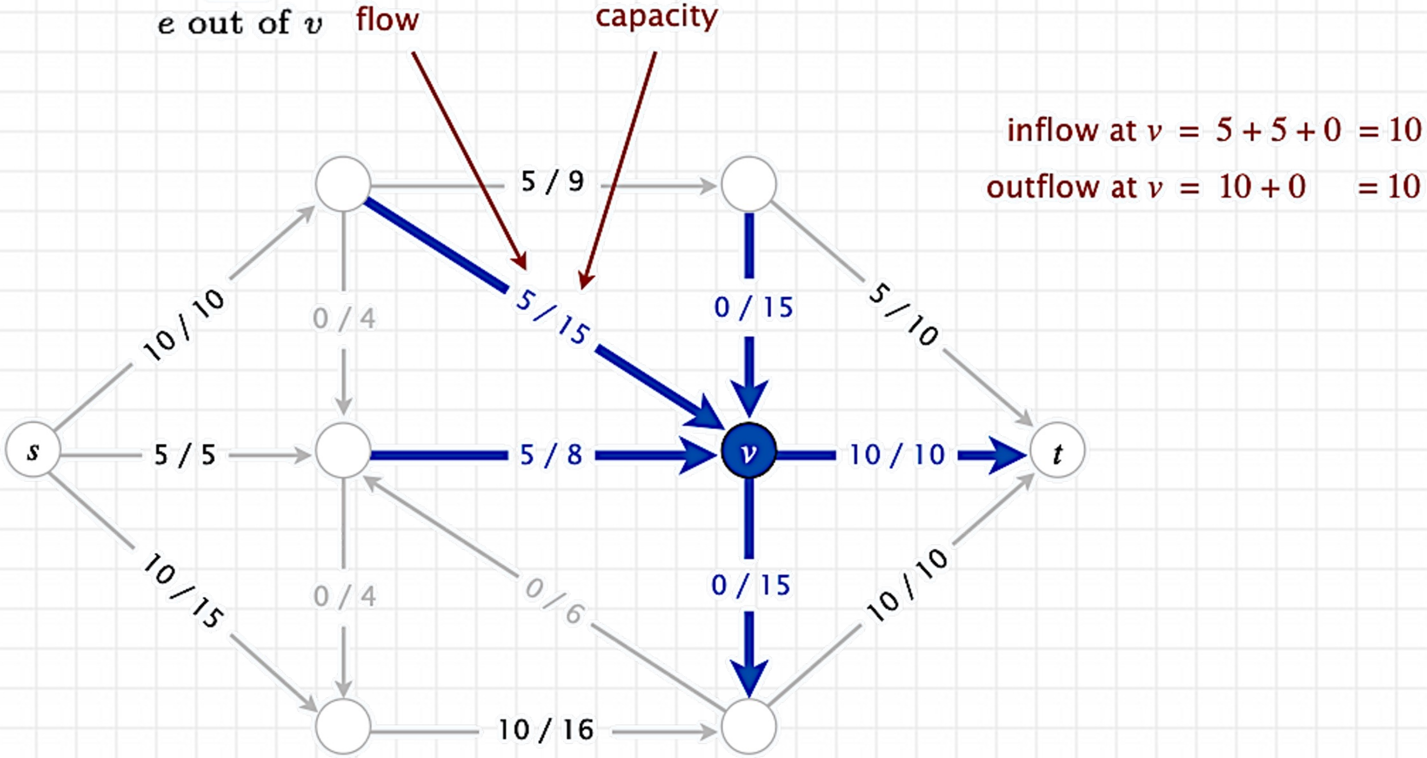
Def. An *st*-flow (flow) f is a function that satisfies:

- 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]

Flow Network: Max-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]



Flow Network: Max-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]

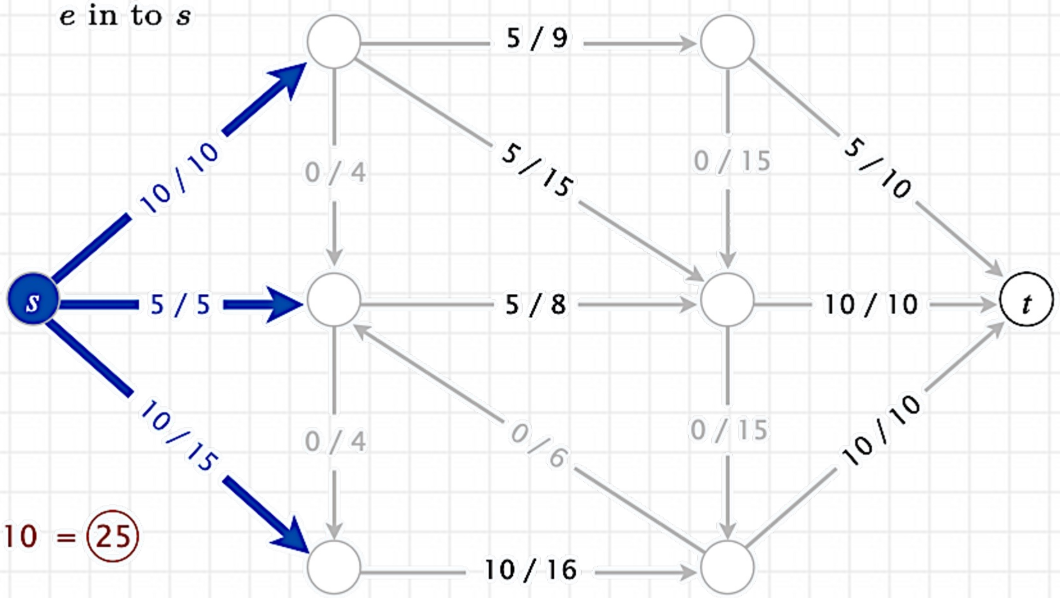
Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

Flow Network: Max-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 $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [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

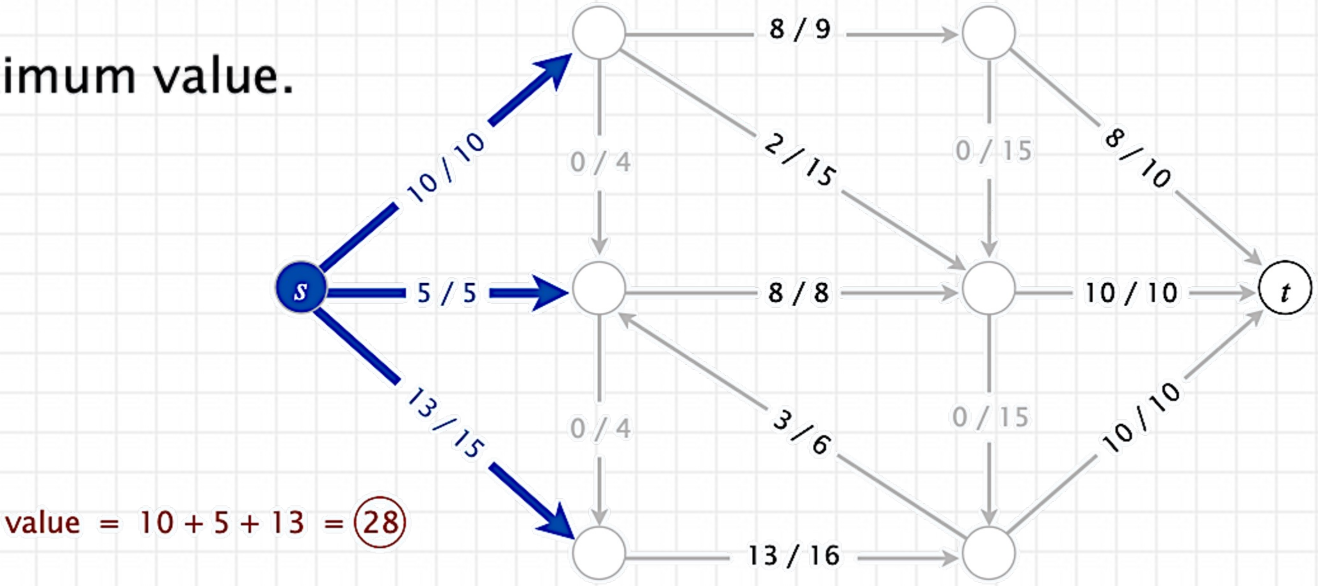
Flow Network: Max-Flow Problem

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. Find a flow of maximum value.



Ford–Fulkerson Algorithm

- Toward a max-flow algorithm

Greedy algorithm.

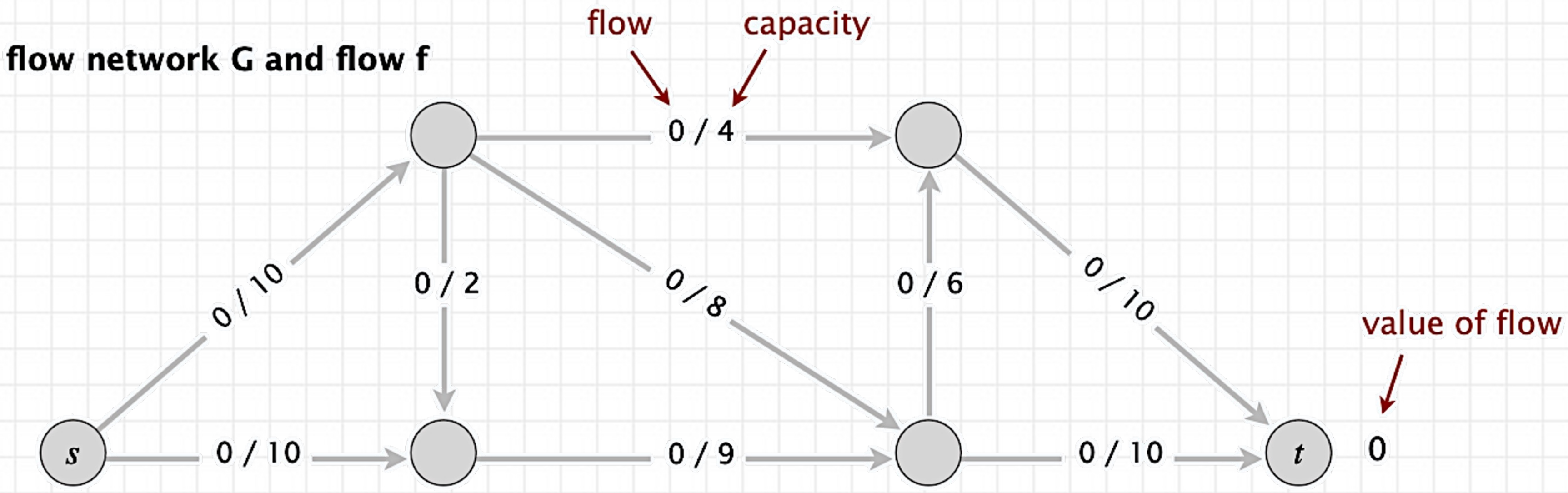
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

Ford–Fulkerson Algorithm

- Toward 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.



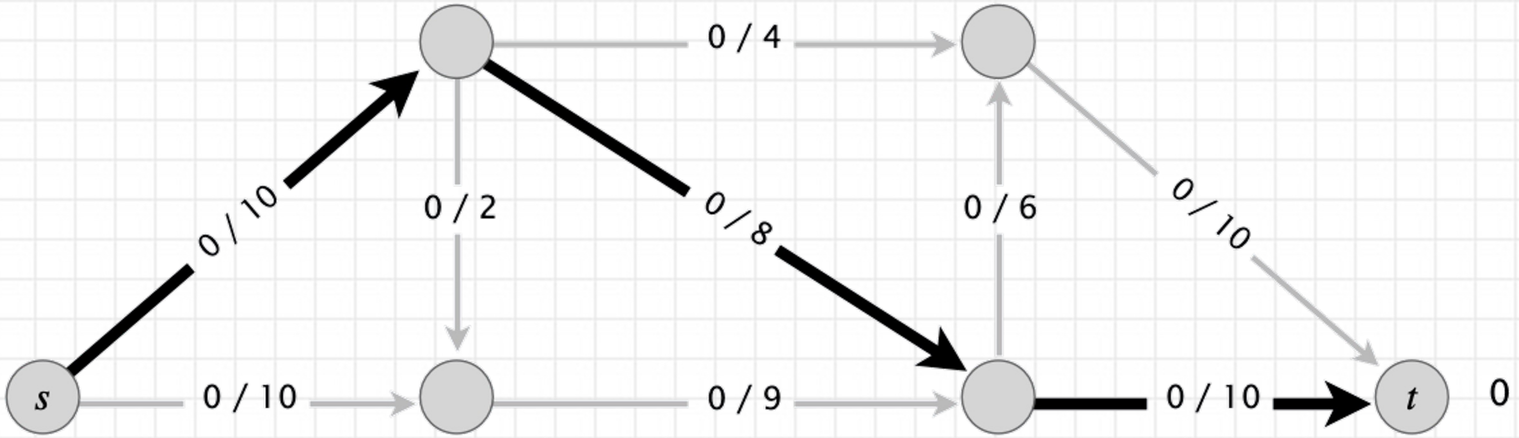
Ford–Fulkerson Algorithm

- Toward 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

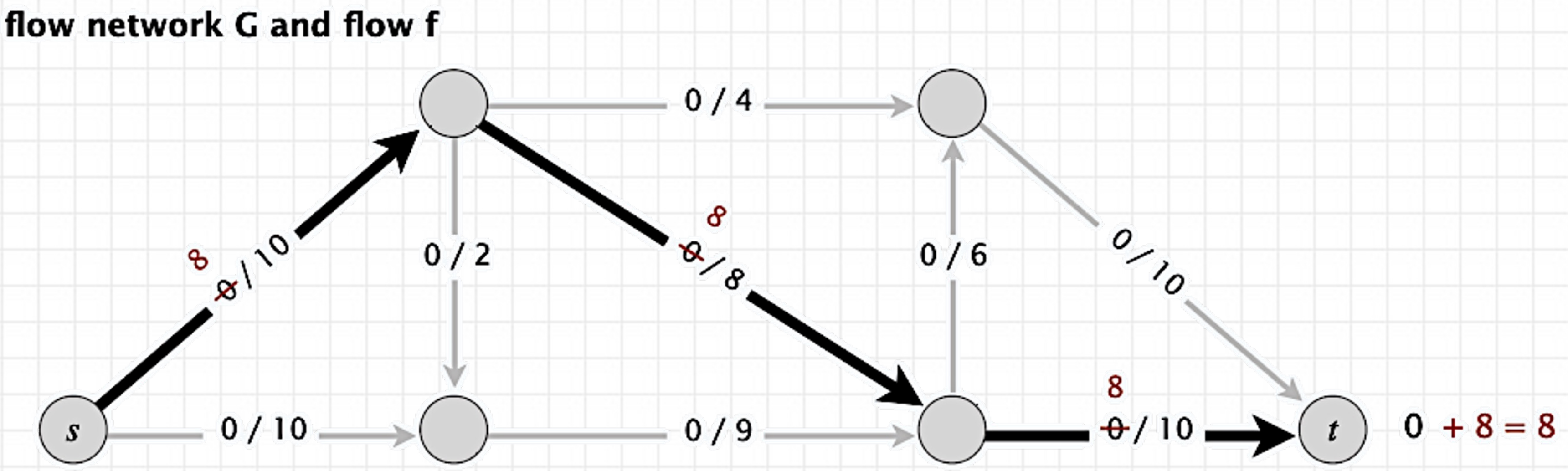


Ford–Fulkerson Algorithm

- Toward 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.



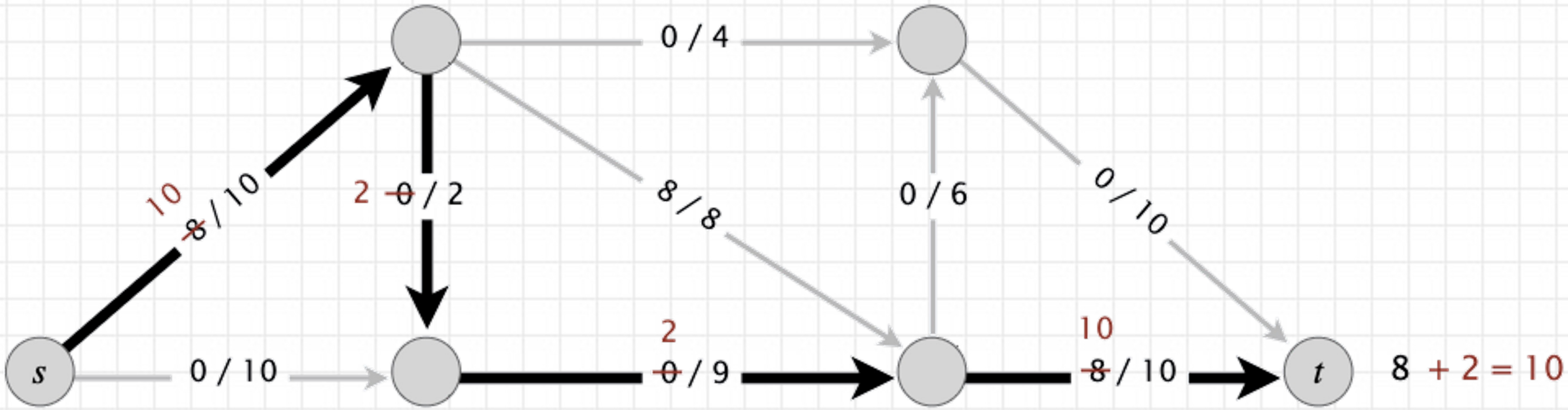
Ford–Fulkerson Algorithm

- Toward 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



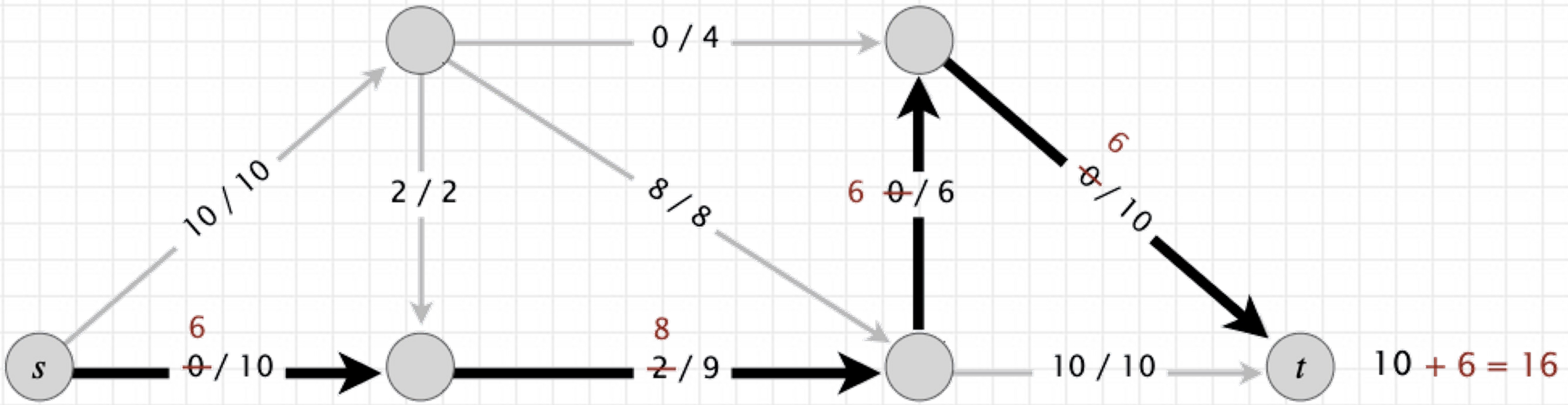
Ford–Fulkerson Algorithm

- Toward 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



Ford–Fulkerson Algorithm

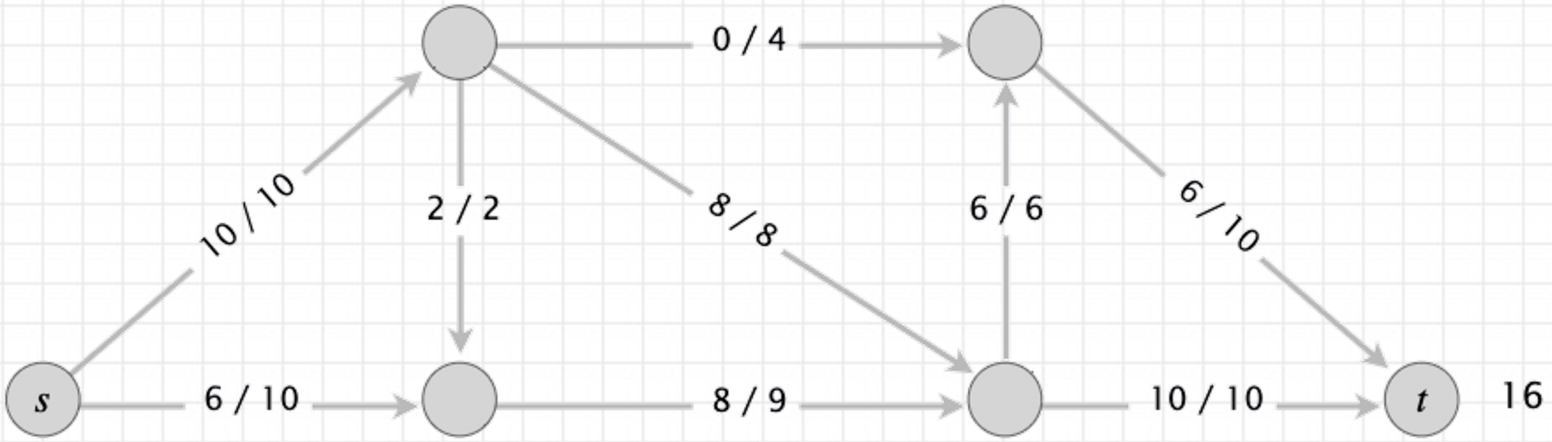
- Toward a max-flow algorithm

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



Ford–Fulkerson Algorithm

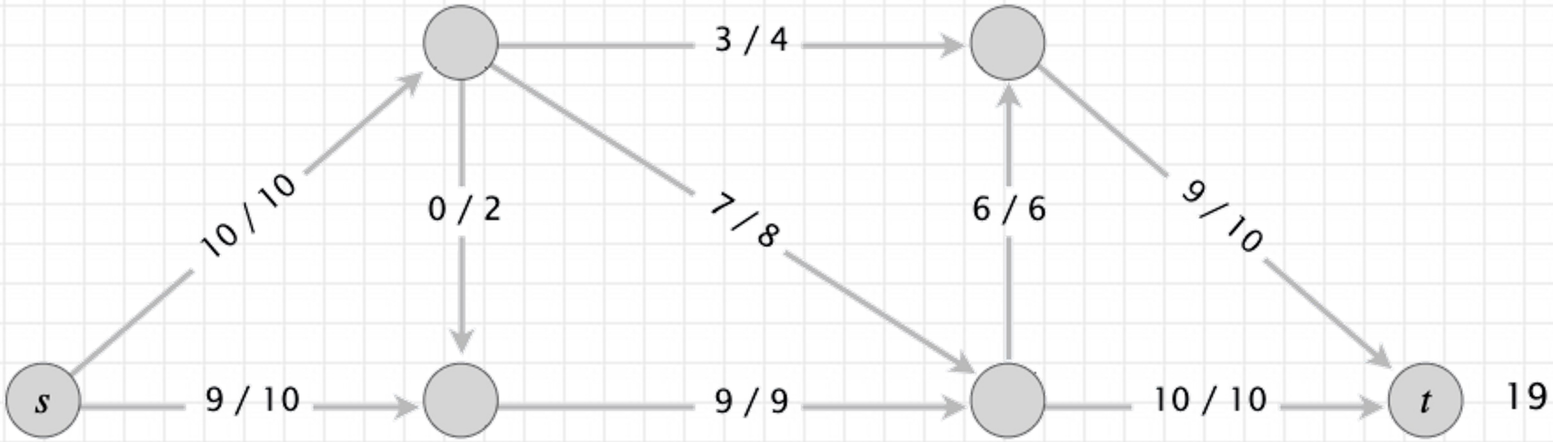
- Toward a max-flow algorithm

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.

but max-flow value = 19

flow network G and flow f

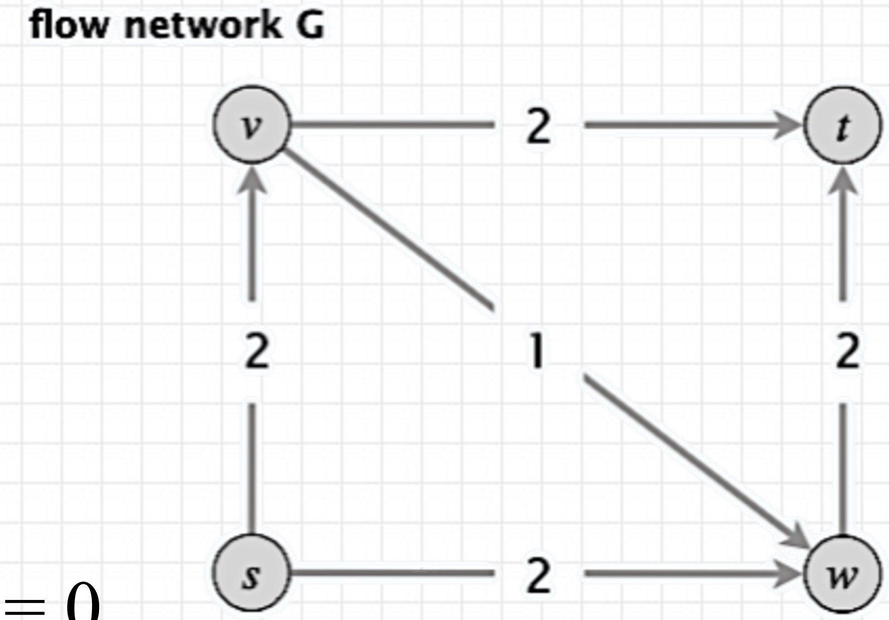


Ford–Fulkerson Algorithm

- Q. Why does the greedy algorithm fail?
- A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ford–Fulkerson Algorithm

- Q. Why does the greedy algorithm fail?
- A. Once greedy algorithm increases flow on an edge, it never decreases it.

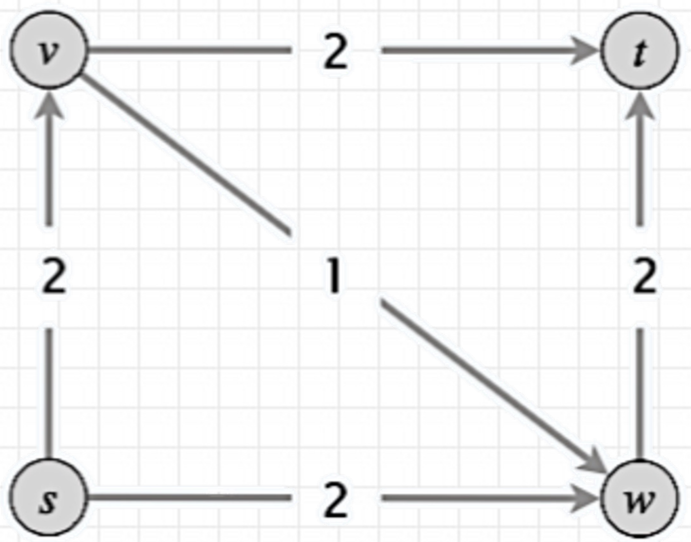


- Ex.
  Consider flow network G .
  The unique max flow f^* has $f^*(v, w) = 0$.
  Greedy algorithm could choose $s \to v \to w \to t$ as first path.

Ford–Fulkerson Algorithm

- Q. Why does the greedy algorithm fail?
- A. Once greedy algorithm increases flow on an edge, it never decreases it.
- <u>Bottom line.</u>
  Need some mechanism to "undo" a bad decision.

flow network G



- Ex.
  Consider flow network G .
  The unique max flow f^* has $f^*(v, w) = 0$.
  Greedy algorithm could choose $s \to v \to w \to t$ as first path.

Residual Network

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

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

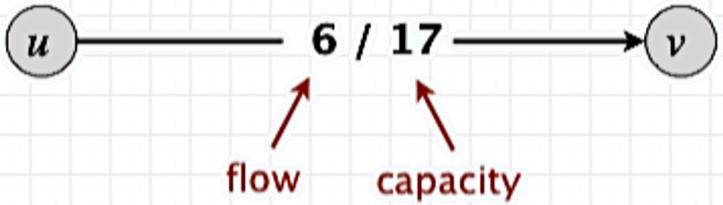
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

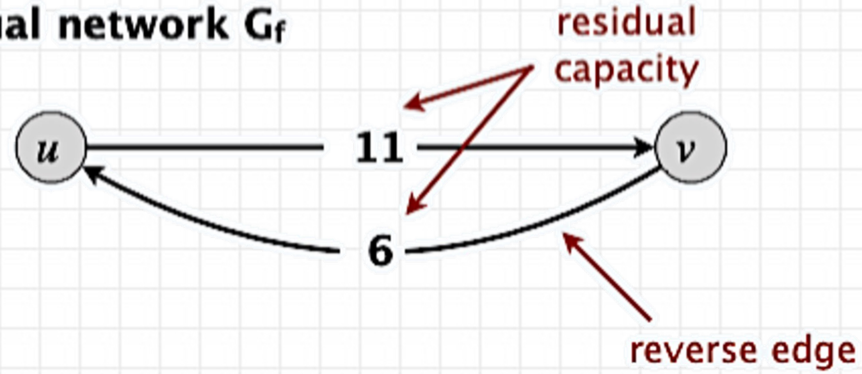
Residual capacity.

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

original flow network G



residual network G_f



Residual Network

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

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

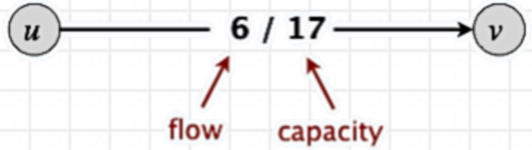
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

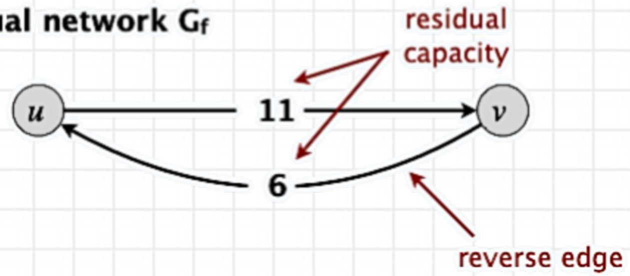
Residual capacity.

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

original flow network G



residual network G_f



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

- $E_f = \{e : f(e) < c(e)\} \cup \{e : f(e^{\text{reverse}}) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

edges with positive residual capacity

where flow on a reverse edge negates flow on corresponding forward edge

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 $val(f') = val(f) + bottleneck(G_f, P)$.

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)

δ ← bottleneck capacity of augmenting path P.
FOREACH edge e ∈ P :
    IF (e ∈ E) f(e) ← f(e) + δ.
    ELSE       f(e^reverse) ← f(e^reverse) – δ.
RETURN f.
```

Ford–Fulkerson Algorithm

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

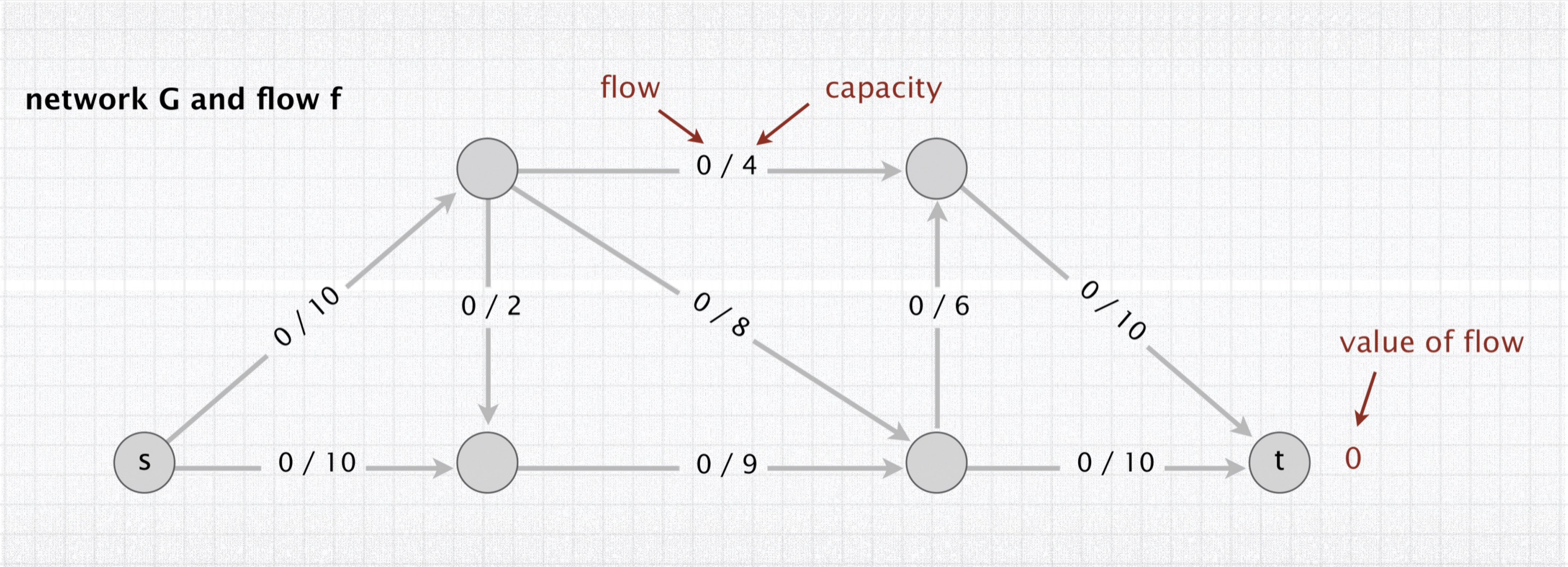
```
FORD–FULKERSON(G)

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

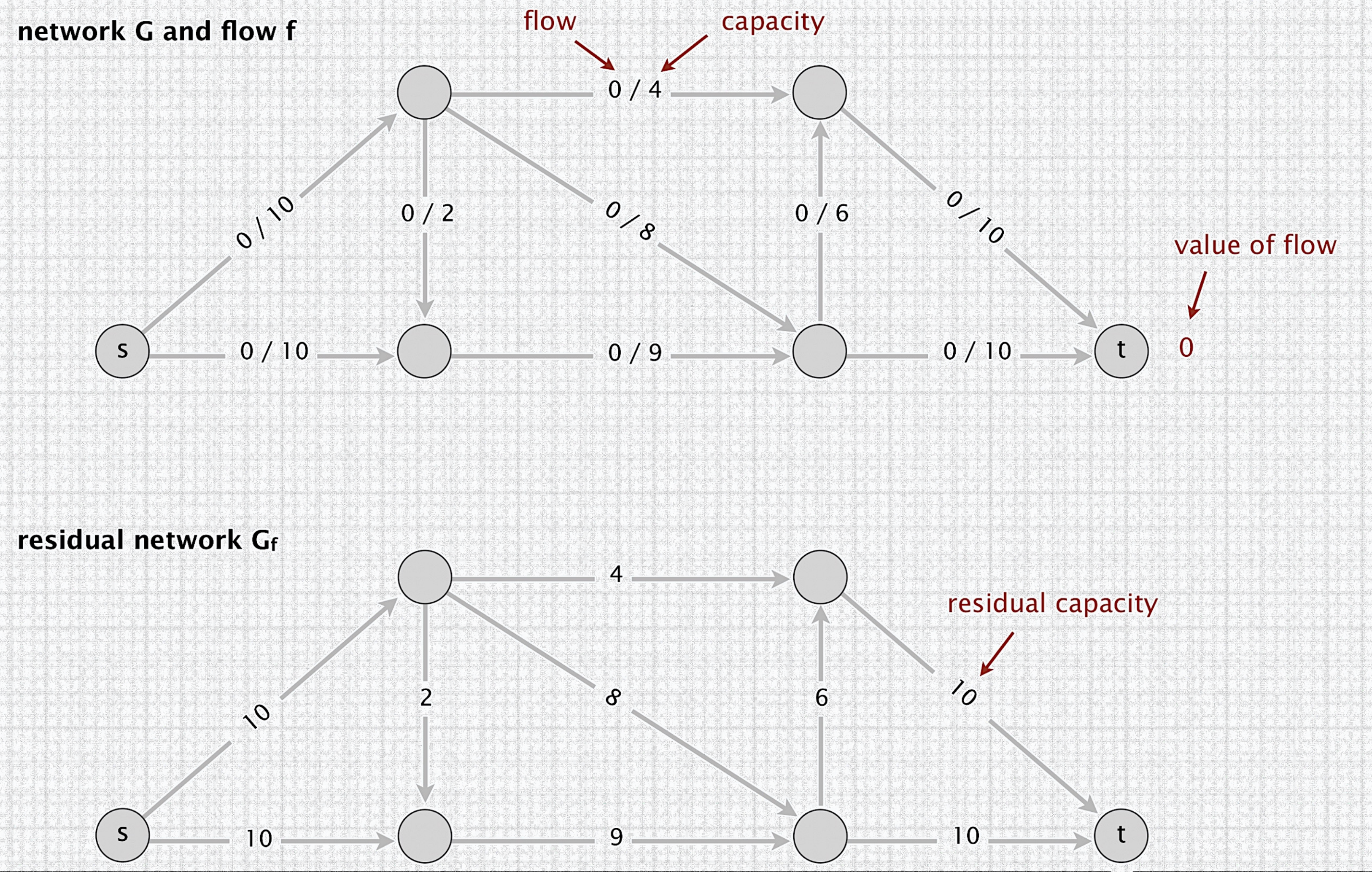
augmenting path

Ford–Fulkerson Algorithm

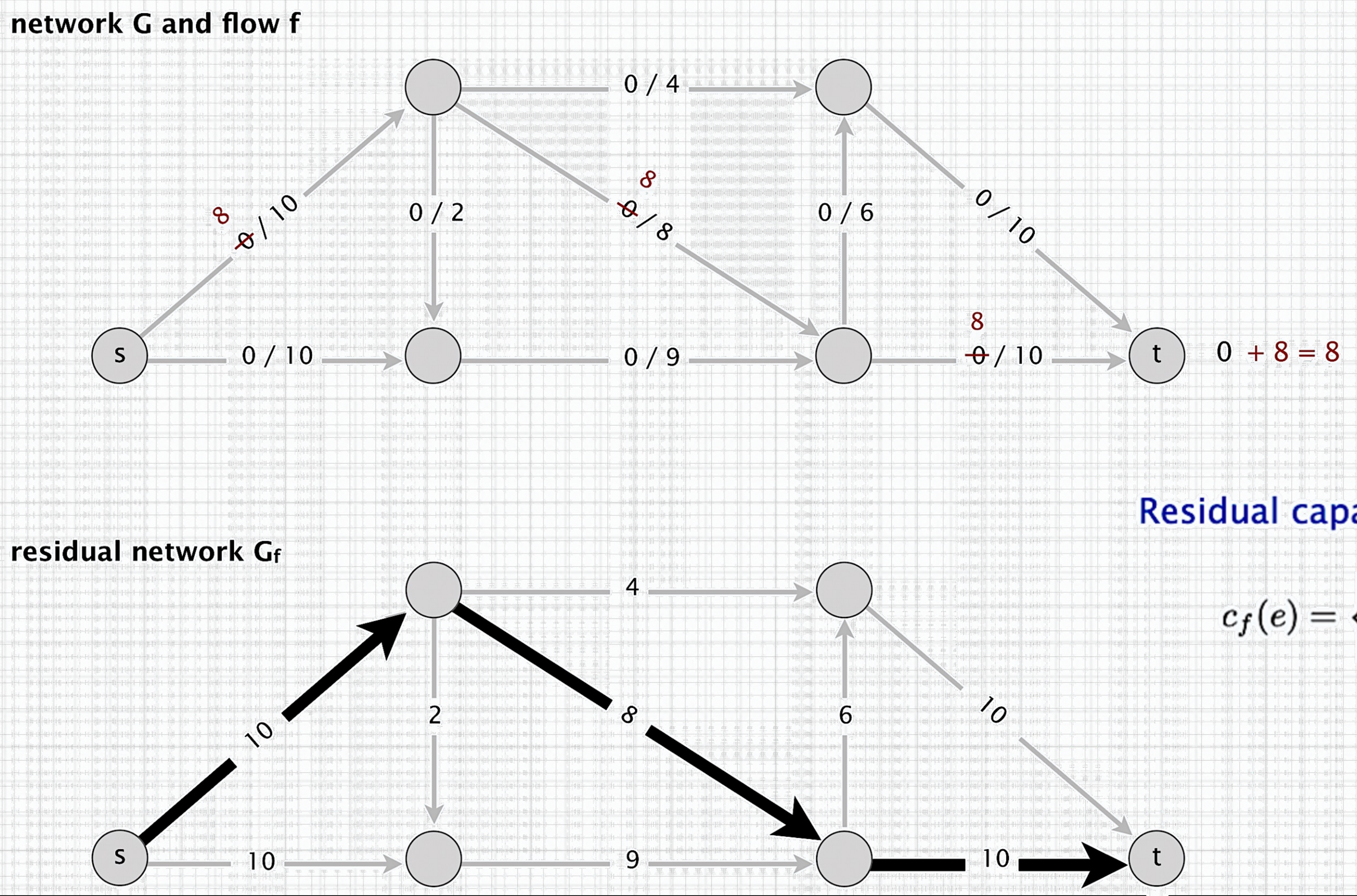
- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P in the residual network G_f .
- Augment flow along path P.
- Repeat until you get stuck.



Ford–Fulkerson Algorithm



Ford–Fulkerson Algorithm

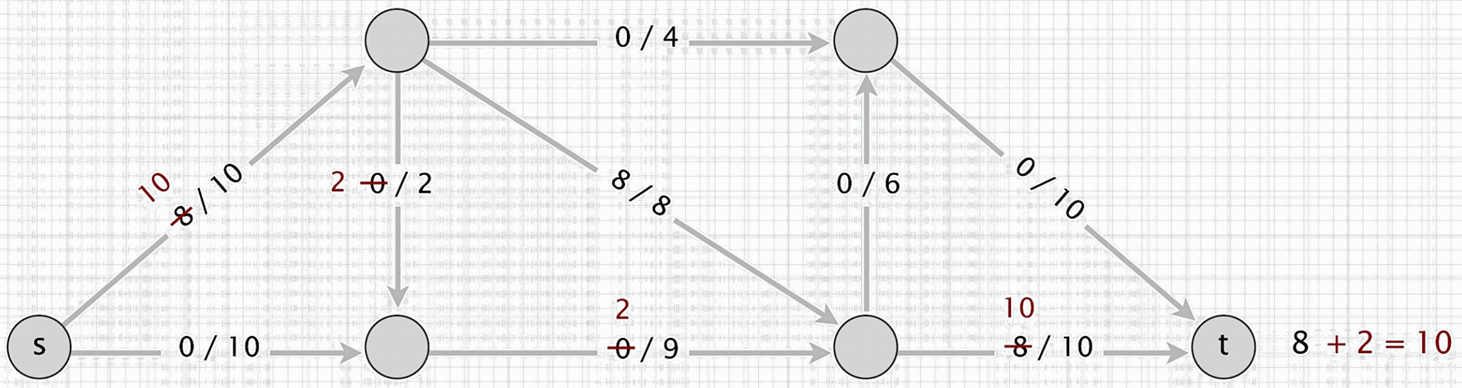


Residual capacity.

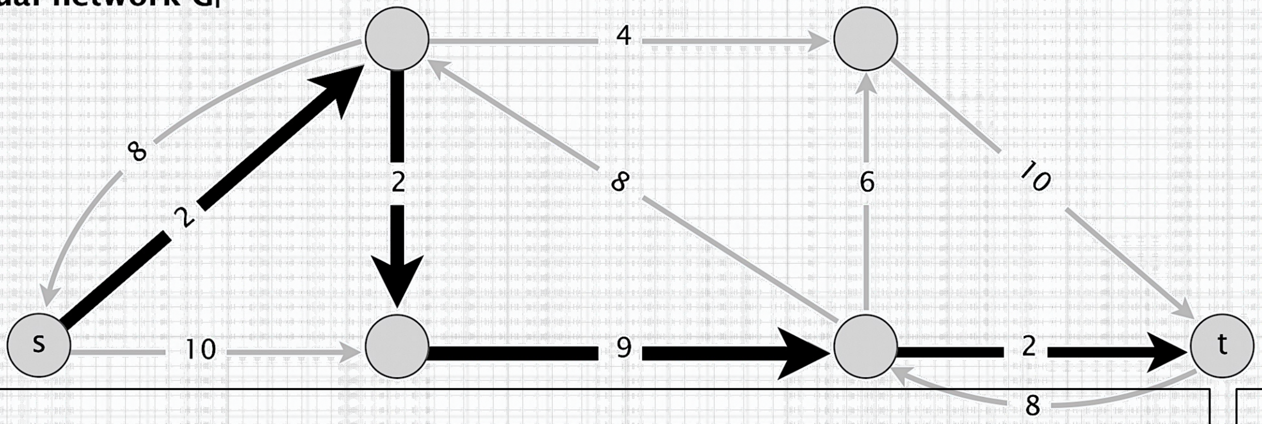
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e^{\text{reverse}}) & \text{if } e^{\text{reverse}} \in E \end{cases}$$

Ford–Fulkerson Algorithm

network G and flow f



residual network G_f

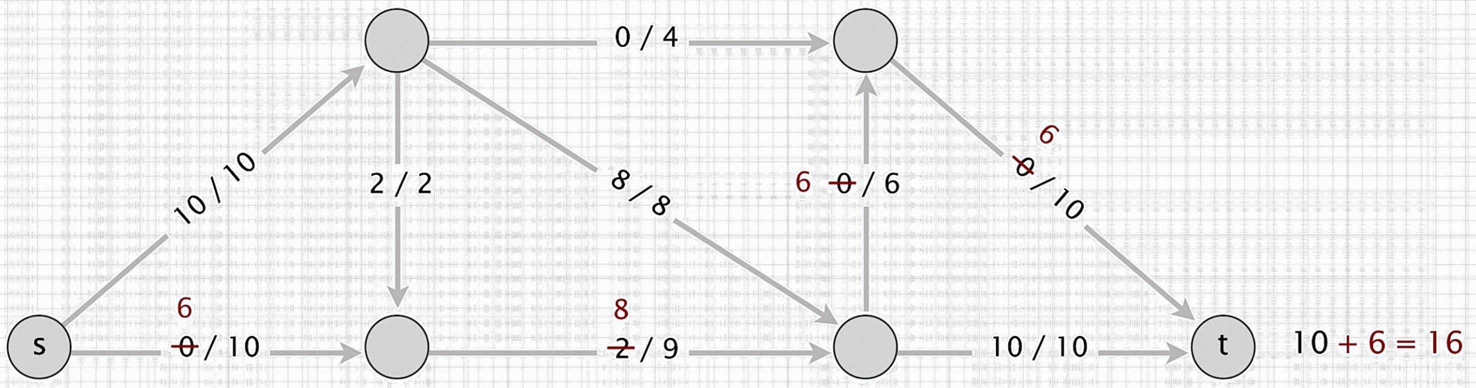


Residual capacity.

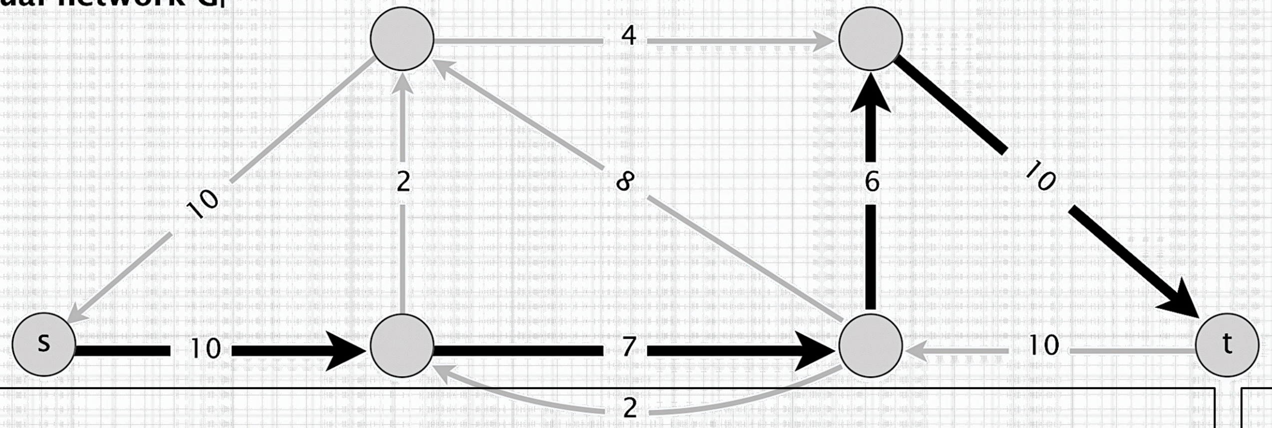
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e^{\text{reverse}}) & \text{if } e^{\text{reverse}} \in E \end{cases}$$

Ford–Fulkerson Algorithm

network G and flow f



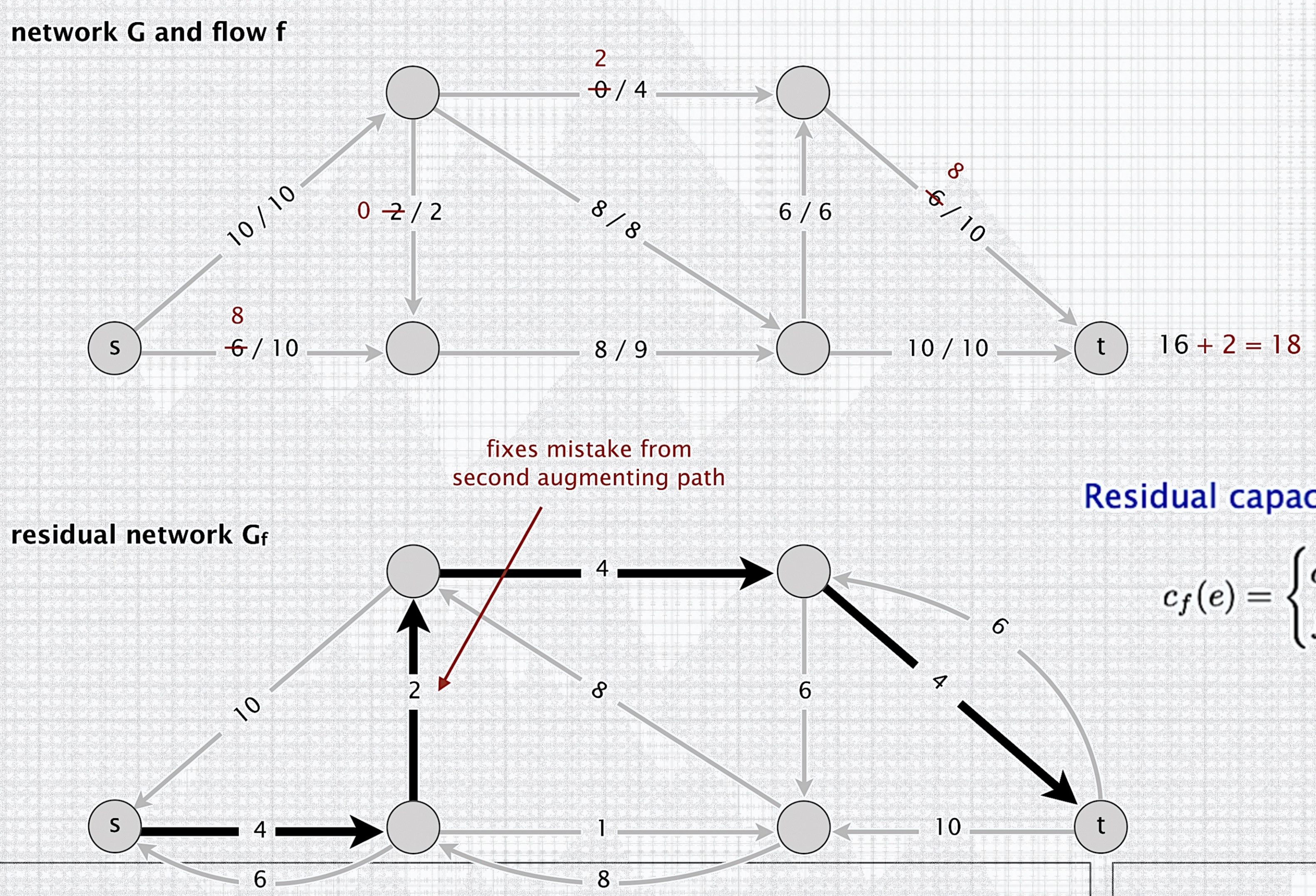
residual network G_f



Residual capacity.

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

Ford–Fulkerson Algorithm

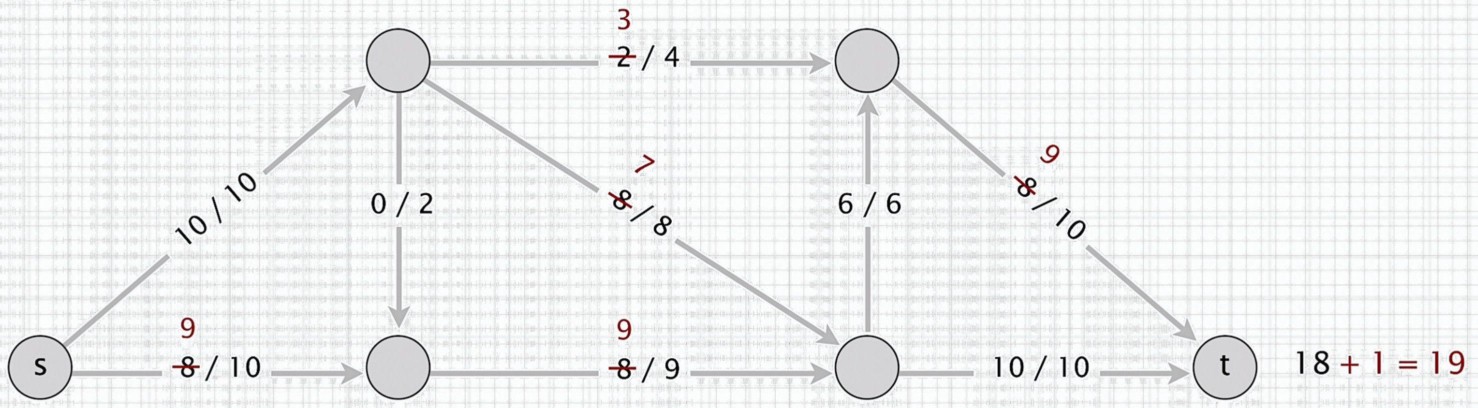


Residual capacity.

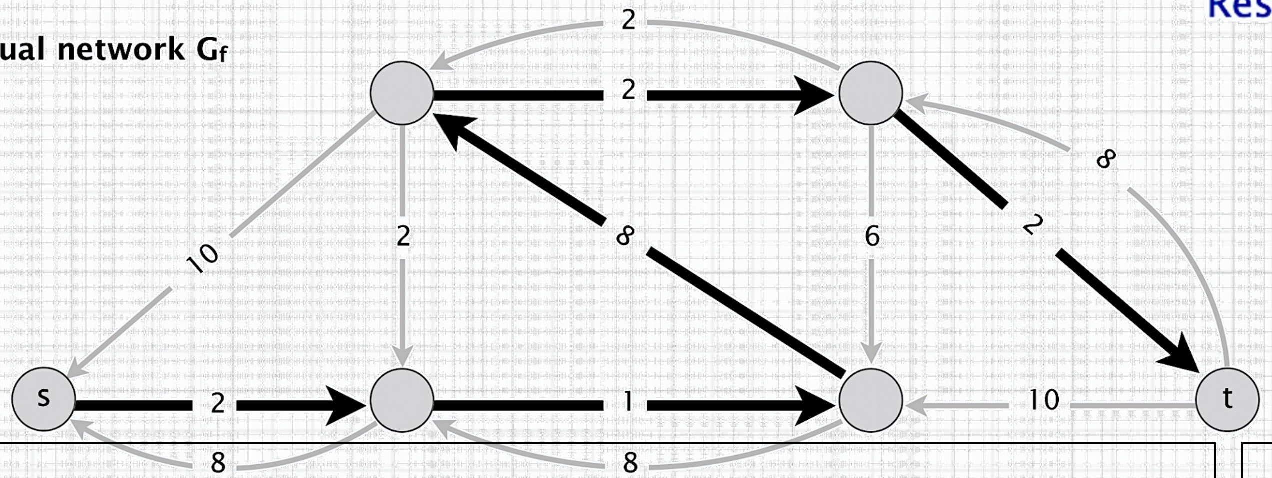
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e^{\text{reverse}}) & \text{if } e^{\text{reverse}} \in E \end{cases}$$

Ford–Fulkerson Algorithm

network G and flow f



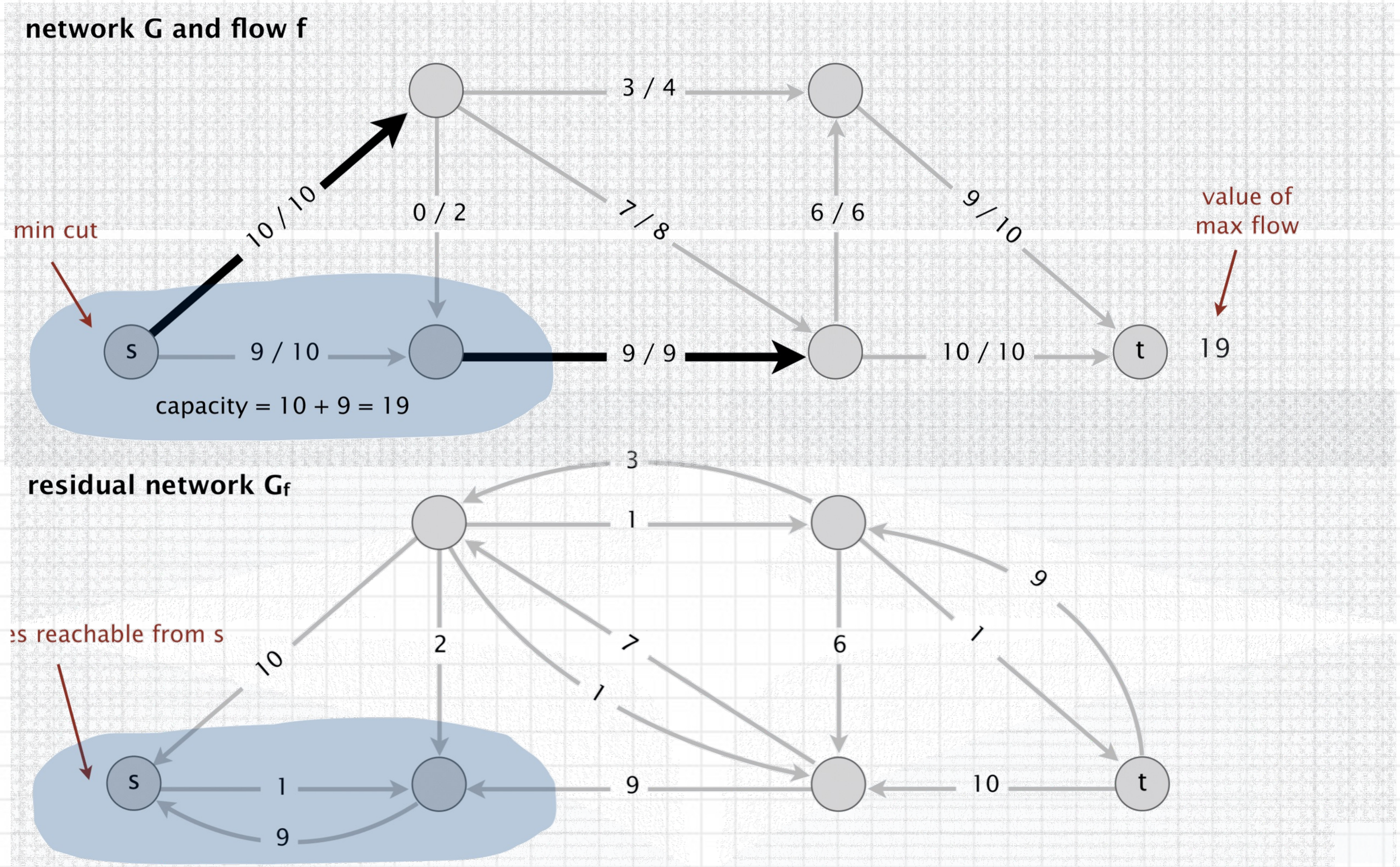
residual network G_f



Residual capacity.

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

Ford–Fulkerson Algorithm



Max-Flow Min-Cut Theorem

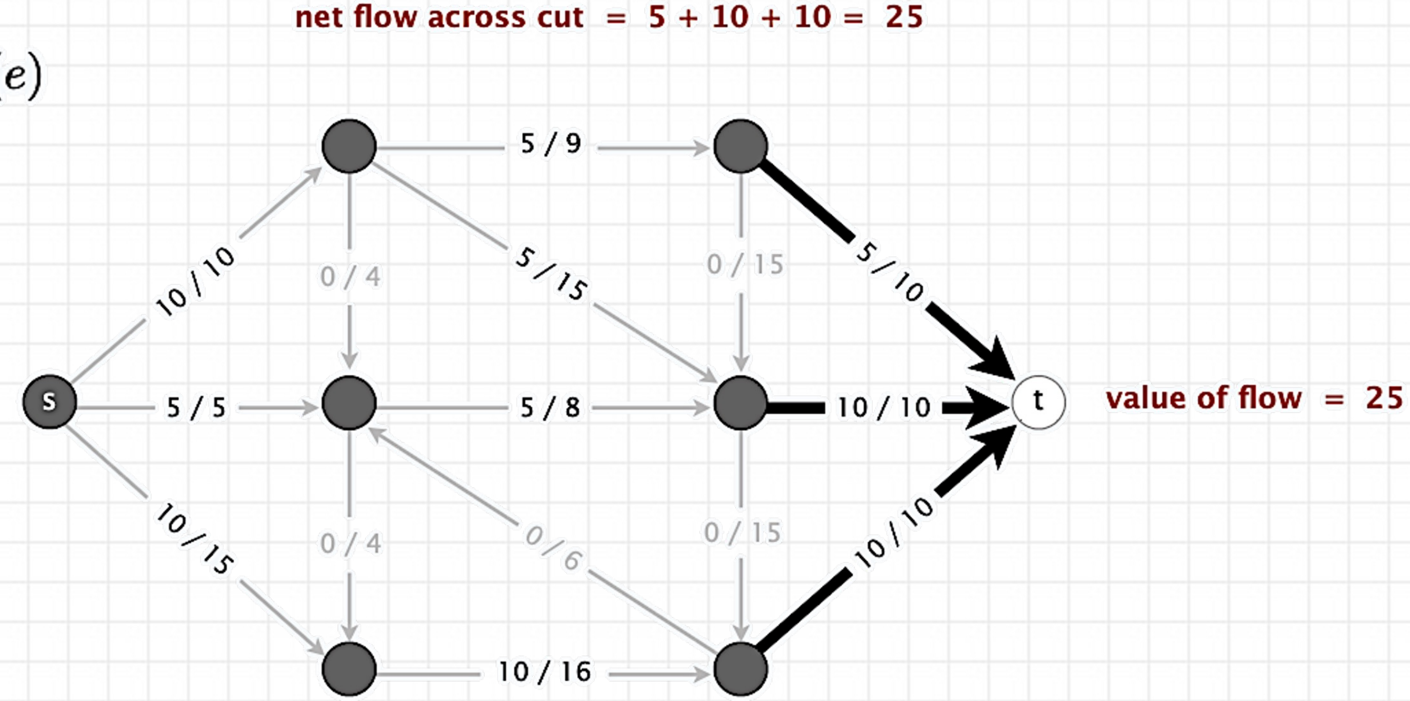
- Relationship between flows and cuts
- Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

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

Max-Flow Min-Cut Theorem

- 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).

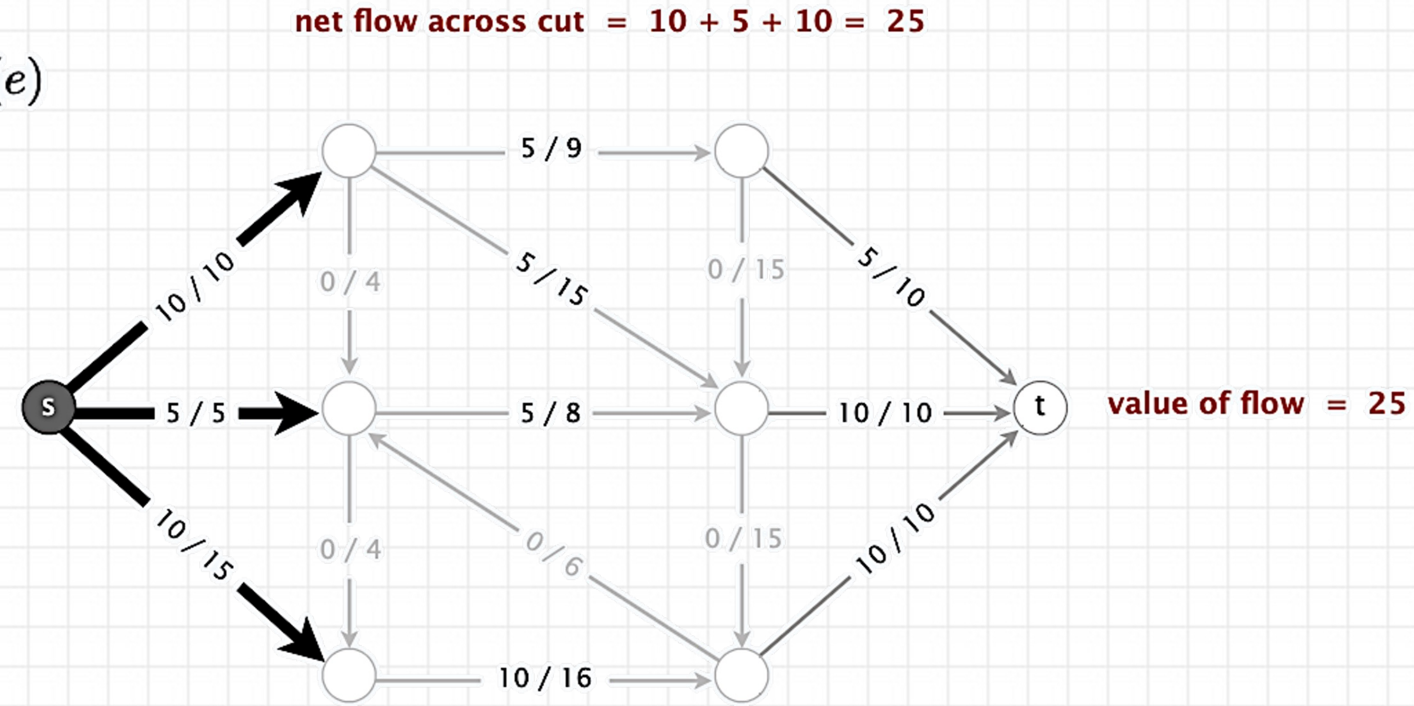
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Max-Flow Min-Cut Theorem

- 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).

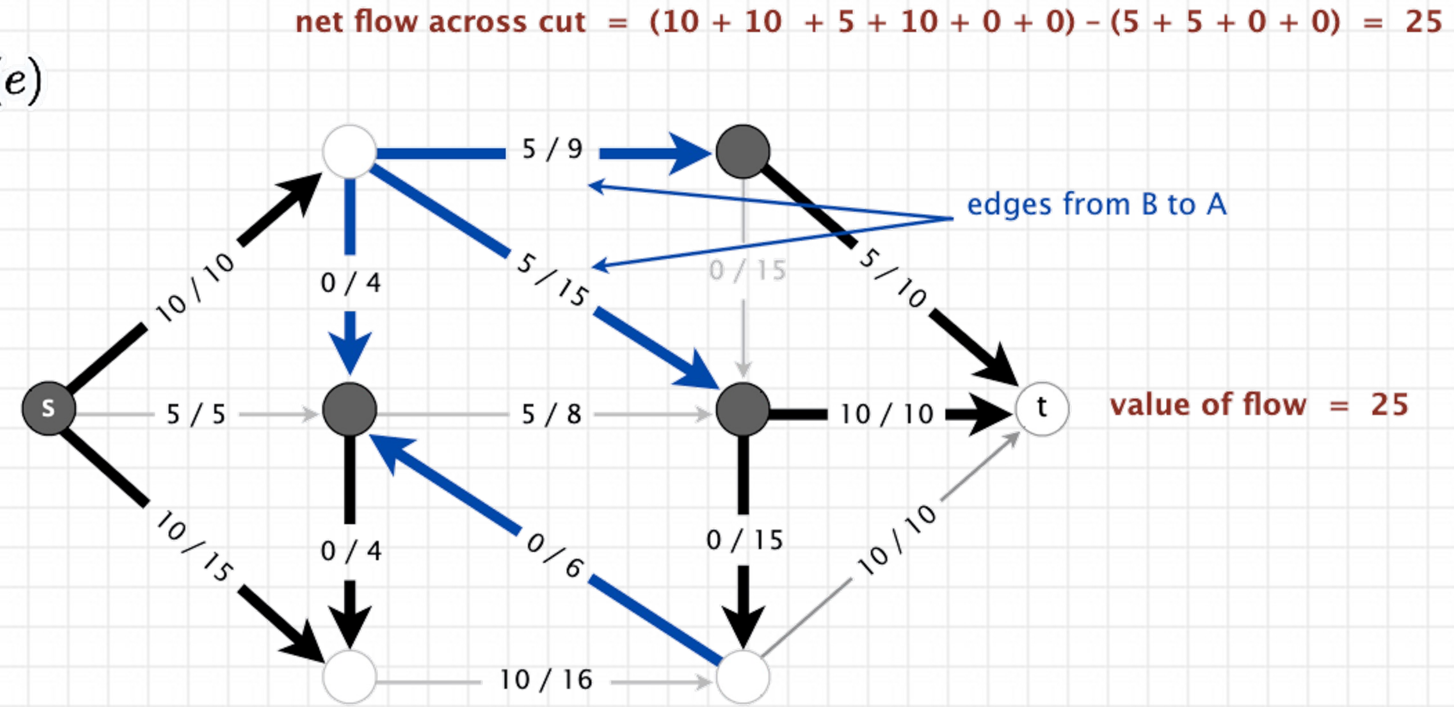
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Max-Flow Min-Cut Theorem

- 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).

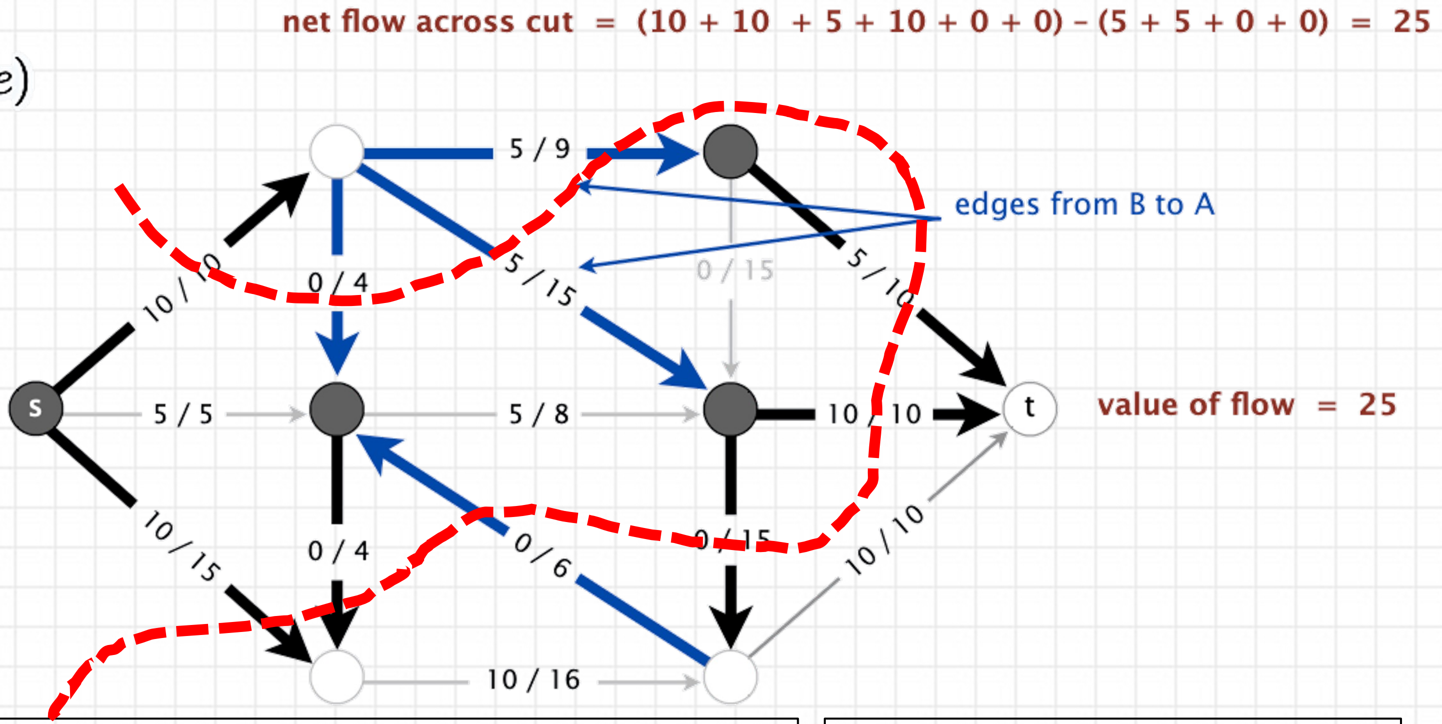
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Max-Flow Min-Cut Theorem

- 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).

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



Max-Flow Min-Cut Theorem

- 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*).

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

- Proof.

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

by flow conservation, all terms except for $v = s$ are 0 →

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

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

Max-Flow Min-Cut Theorem

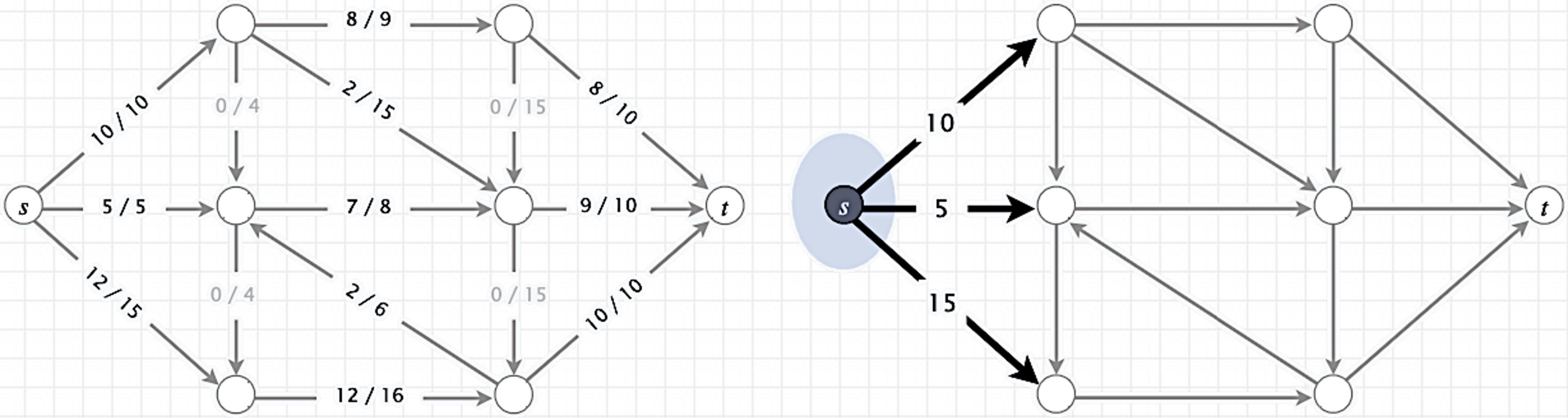
- Relationship between flows and cuts
- Weak duality. Let f be any flow and (A, B) be any cut. Then, $val(f) \leq cap(A, B)$.
- Proof.

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

flow value lemma

Max-Flow Min-Cut Theorem

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



value of flow = 27 $\leq$ capacity of cut = 30

Max-Flow Min-Cut Theorem

- Relationship between flows and cuts
- 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.

- Proof.

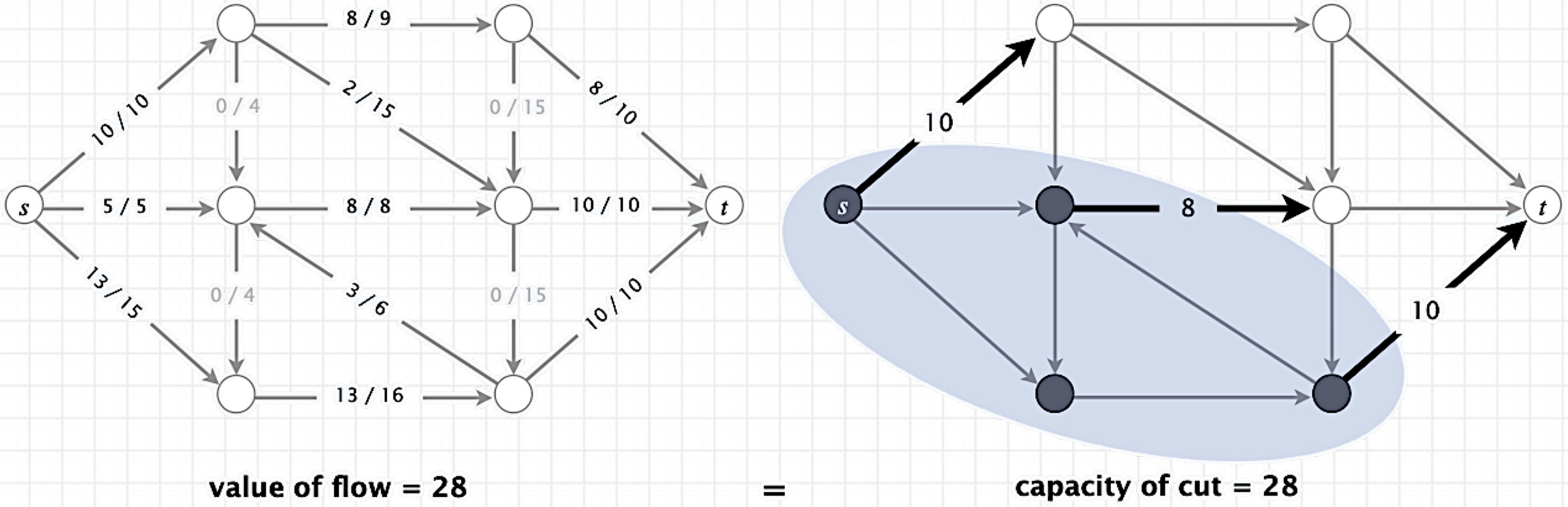
  weak duality

  - For any flow f': $val(f') \leq cap(A, B) = val(f)$.
  - For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$. ▪

  weak duality

Max-Flow Min-Cut Theorem

- Relationship between flows and cuts
- 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.



Max-Flow Min-Cut Theorem

- Max-flow min-cut theorem

Value of a max flow = capacity of a min cut

strong duality

Max-Flow Min-Cut Theorem

- Max-flow min-cut theorem: Value of a max flow = capacity of a min cut
- Augmenting path theorem: A flow f is a max flow iff no augmenting paths.
- Proof : The following three conditions are equivalent for any flow f :
  1. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
  2. f is a max flow.
  3. There is no augmenting path with respect to f.

Max-Flow Min-Cut Theorem

- Max-flow min-cut theorem: Value of a max flow = capacity of a min cut
- Augmenting path theorem: A flow f is a max flow iff no augmenting paths.
- Proof : The following three conditions are equivalent for any flow f :
  1. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
  2. f is a max flow.
  3. There is no augmenting path with respect to f.

if Ford–Fulkerson terminates, then f is max flow

Max-Flow Min-Cut Theorem

- Max-flow min-cut theorem: Value of a max flow = capacity of a min cut
- Augmenting path theorem: A flow f is a max flow iff no augmenting paths.
- Proof : The following three conditions are equivalent for any flow f :
  1. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
  2. f is a max flow.
  3. There is no augmenting path with respect to f.

if Ford–Fulkerson terminates, then f is max flow

- 1⇒2
  - This is the weak duality corollary.

Max-Flow Min-Cut Theorem

- Max-flow min-cut theorem: Value of a max flow = capacity of a min cut
- Augmenting path theorem: A flow f is a max flow iff no augmenting paths.
- Proof : The following three conditions are equivalent for any flow f :
  1. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
  2. f is a max flow.
  3. There is no augmenting path with respect to f.

- 2⇒3 We prove contrapositive: $\neg 3 \Rightarrow \neg 2$.
  - 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

- Max-flow min-cut theorem: Value of a max flow = capacity of a min cut
- Augmenting path theorem: A flow f is a max flow iff no augmenting paths.
- Proof : The following three conditions are equivalent for any flow f :
  1. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
  2. f is a max flow.
  3. There is no augmenting path with respect to f.

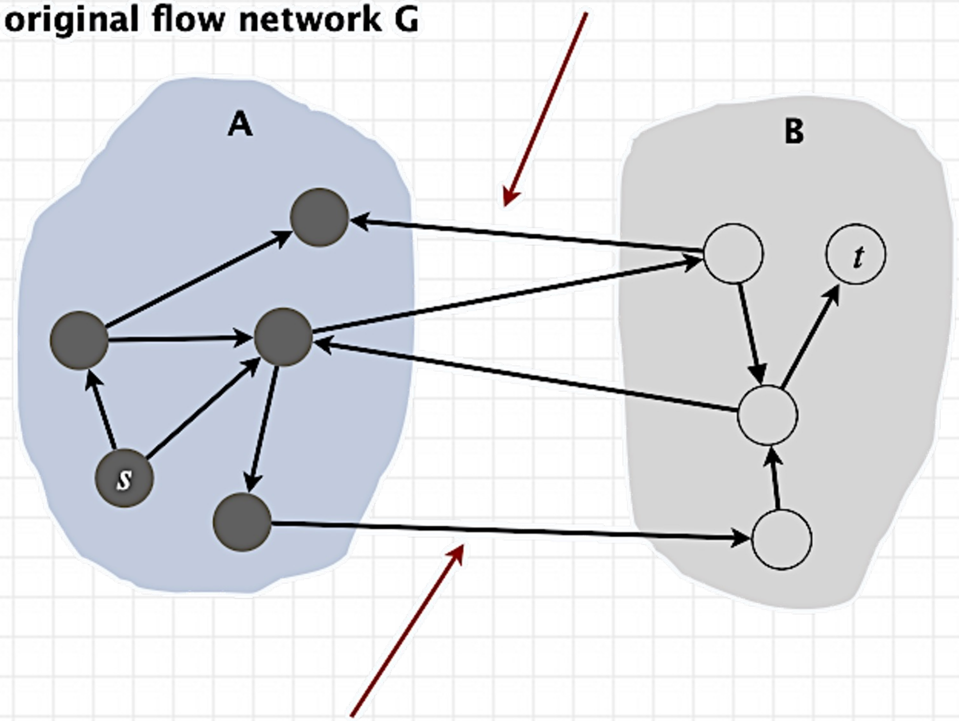
- 3⇒1
  - Let f be a flow with no augmenting paths.
  - Let A = set of nodes reachable from s in residual network G_f.
  - By definition of A: $s \in A$.
  - By definition of flow f: $t \notin A$.

Max-Flow Min-Cut Theorem

- 3⇒1
  - Let f be a flow with no augmenting paths.
  - Let A = set of nodes reachable from s in residual network G_f.
  - By definition of A: $s \in A$.
  - By definition of flow f: $t \notin A$.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&= \sum_{e \text{ out of } A} c(e) - 0 \\
&= cap(A, B) \quad \blacksquare
\end{aligned}
$$

flow value lemma



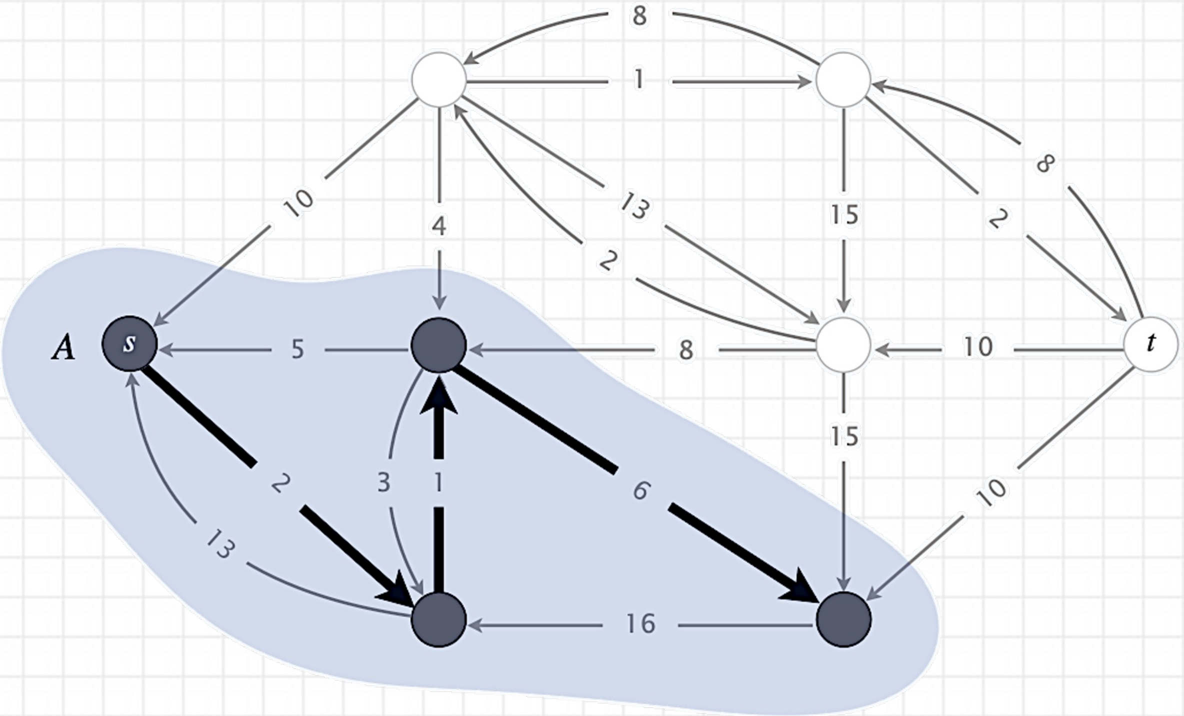
Max-Flow Min-Cut Theorem

- Computing a minimum cut from a maximum flow
- Theorem. Given any max flow f, can compute a min cut (A, B) in $O(|E|)$ time.
- Proof. Let A = set of nodes reachable from s in residual network G_f. ▪

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

Max-Flow Min-Cut Theorem

- Computing a minimum cut from a maximum flow
- Theorem. Given any max flow f, can compute a min cut (A, B) in $O(|E|)$ time.
- Proof. Let A = set of nodes reachable from s in residual network G_f. ▪



Graph

- Graph definition and representation
  - Adjacency matrix
  - Adjacency list
- Graph traversal
  - Breadth first search (BFS)
    - Shortest path (unweighted graphs)
    - Testing bipartiteness
    - Tree traversal (level-order)
    - Connected components
  - Depth first search (DFS)
    - Topological sorting
    - Tree traversal (in-order, pre-order, post-order)
    - Connected components
- Graph problems/algorithms
  - Minimum spanning tree (MST)
    - Kruskal (greedy)
    - Prim (greedy)
  - Shortest path (directed weighted graphs)
    - Dijkstra (greedy)
    - Bellman-Ford (dynamic programming)
    - Floyd-Warshall (dynamic programming)
  - Flow network
    - Max-flow min-cut theorem
    - Ford-Fulkerson algorithm

References

- The lecture slides are mainly based on the suggested textbooks and the corresponding published lecture notes:

  - Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. (Main reference)
  - KT: Kleinberg, J., & Tardos, E. Algorithm design. Pearson/Addison-Wesley, 2006.
  - CLRS: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009.