Network Flow

CS 3AC3

Ryszard Janicki

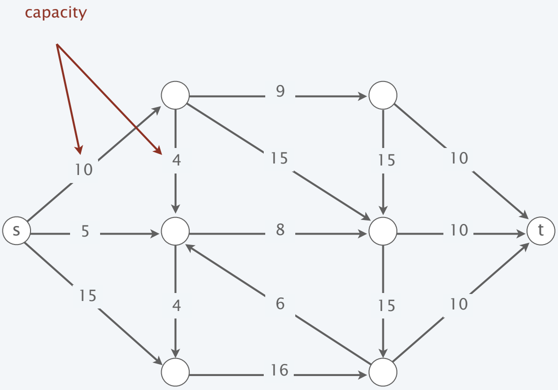
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 7)

Flow network

- Abstraction for material flowing through the edges.
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$.
- Nonnegative integer capacity $c(e)$ for each $e \in E$.

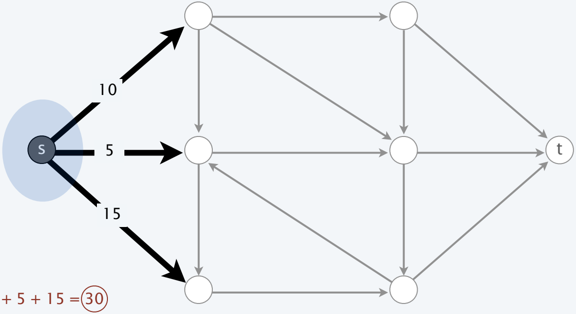
no parallel edges
no edge enters s
no edge leaves t



Def. A *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.

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



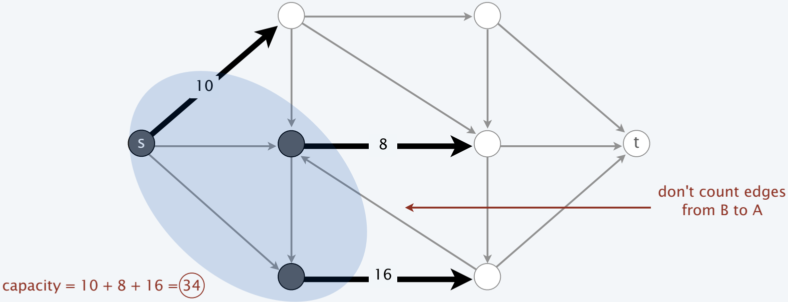
capacity = 10 + 5 + 15 = 30

Minimum cut problem

Def. A *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.

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



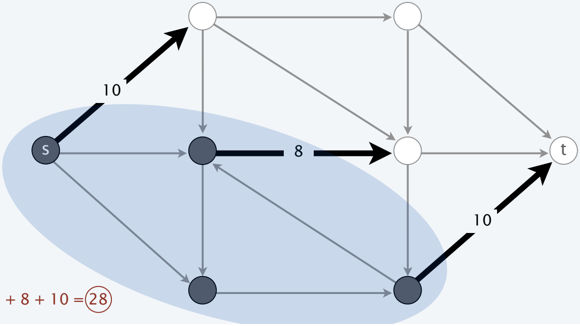
Minimum cut problem

Def. A *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.

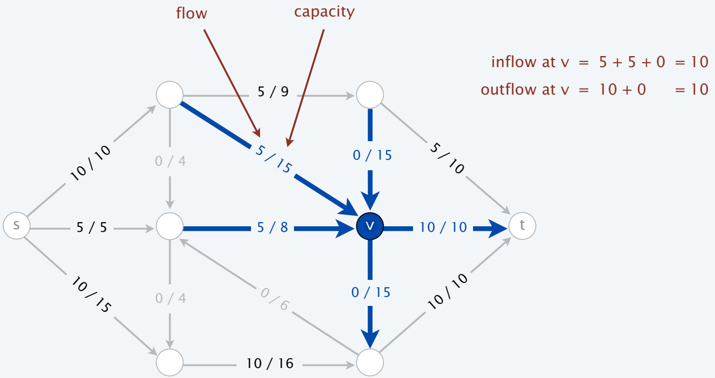


capacity = 10 + 8 + 10 = 28

Maximum flow problem

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

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

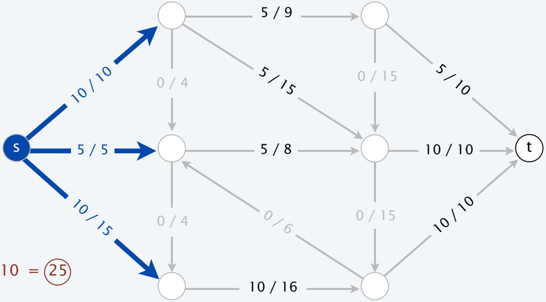


Maximum flow problem

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

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [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: $val(f) = \sum_{e \text{ out of } s} f(e)$.



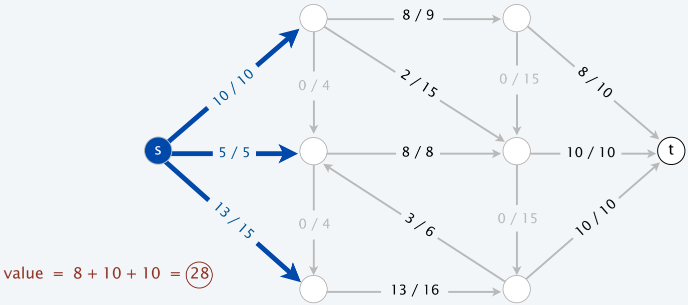
value = 5 + 10 + 10 = 25

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]

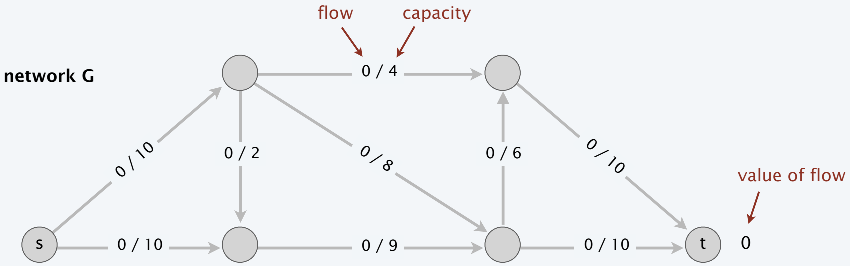
Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e)$.

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



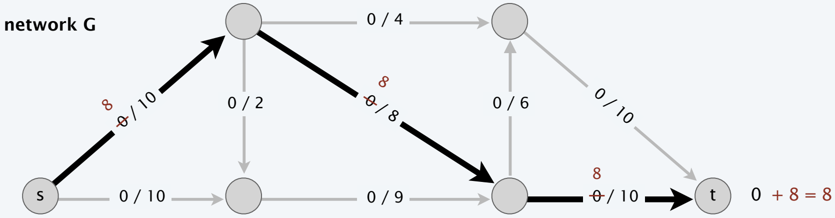
Greedy algorithm.

- Start with $f(e) = 0$ for all 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.



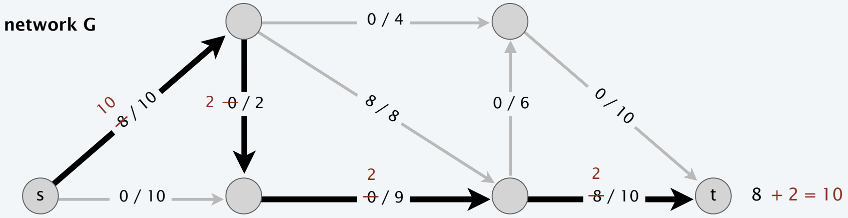
Greedy algorithm.

- Start with $f(e) = 0$ for all 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.



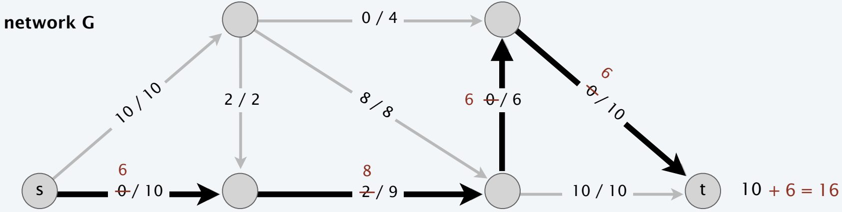
Greedy algorithm.

- Start with $f(e) = 0$ for all 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.



Greedy algorithm.

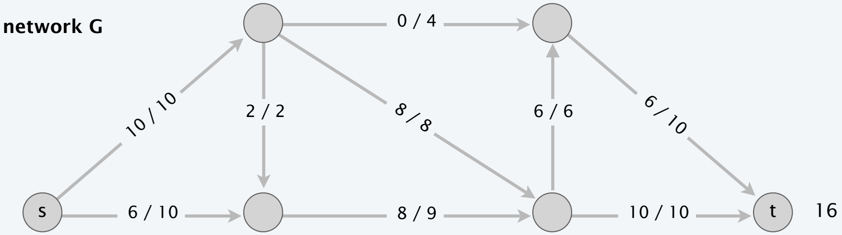
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Greedy algorithm.

- Start with $f(e) = 0$ for all 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

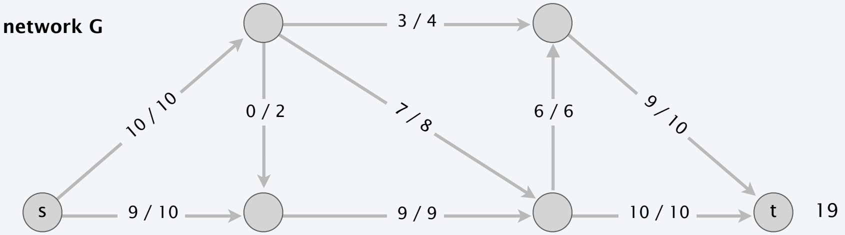


GREEDY DOES NOT WORK!

Greedy algorithm.

- Start with $f(e) = 0$ for all 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.

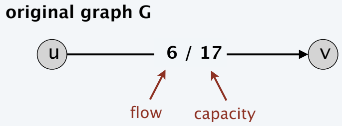
but max-flow value = 19



Residual graph

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

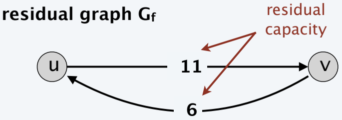
- Flow $f(e)$.
- Capacity $c(e)$.



Residual edge.

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

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



Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

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

Augmenting path

Def. An augmenting path is a simple $s \to t$ path P in the residual graph G_f.

Def. The bottleneck capacity of an augmenting 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 f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

AUGMENT (f, c, P)

$b \leftarrow$ bottleneck capacity of path P.
FOREACH edge $e \in P$
  IF $(e \in E)$ $f(e) \leftarrow f(e) + b$.
  ELSE $f(e^R) \leftarrow f(e^R) - b$.
RETURN f.

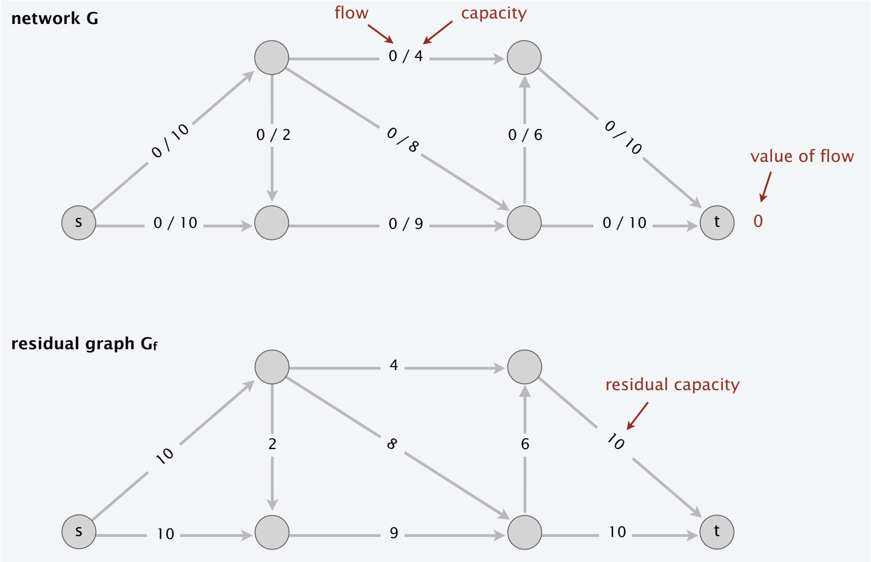
Ford-Fulkerson algorithm

Ford-Fulkerson augmenting path algorithm.

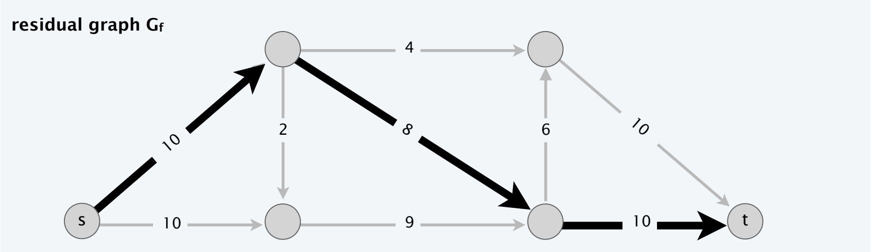
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path P in the residual graph G_f.
- Augment flow along path P.
- Repeat until you get stuck.

```
FORD-FULKERSON (G, s, t, c)
-----------------------------------------------
FOREACH edge e ∈ E : f(e) ← 0.
   G_f ← residual graph.
   WHILE (there exists an augmenting path P in G_f)
      f ← AUGMENT (f, c, P).
      Update G_f.
   RETURN f.
}
```

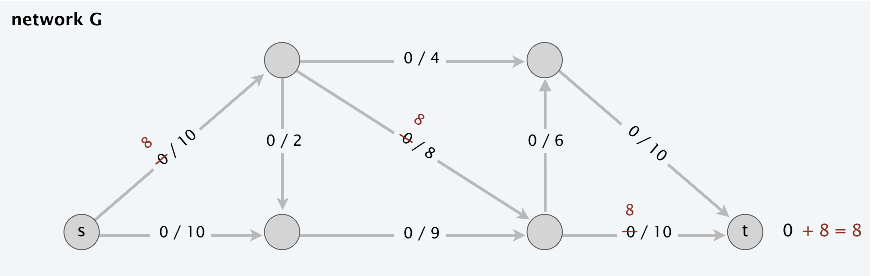
Ford-Fulkerson algorithm demo



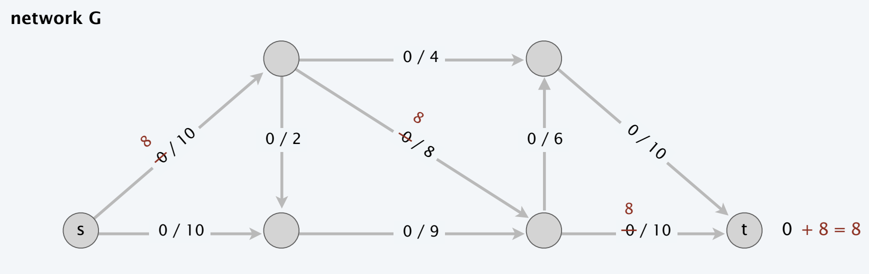
Ford-Fulkerson algorithm demo



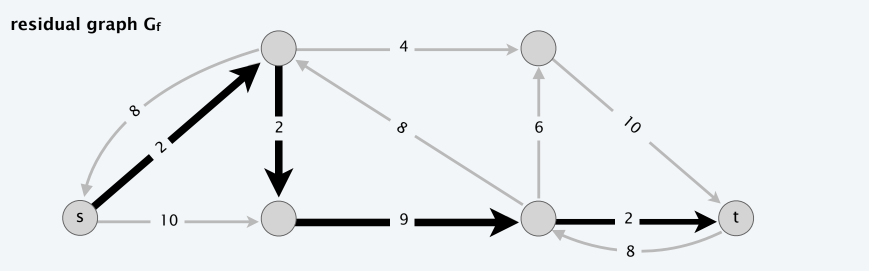
bottleneck = 8 ⇓



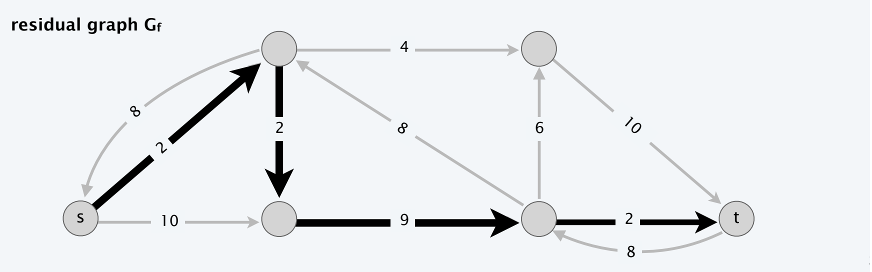
Ford-Fulkerson algorithm demo



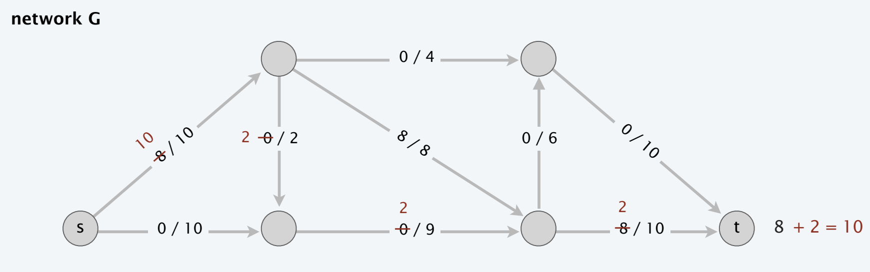
⇓



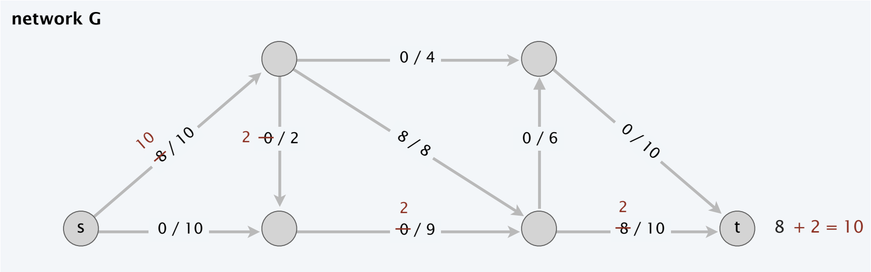
Ford-Fulkerson algorithm demo



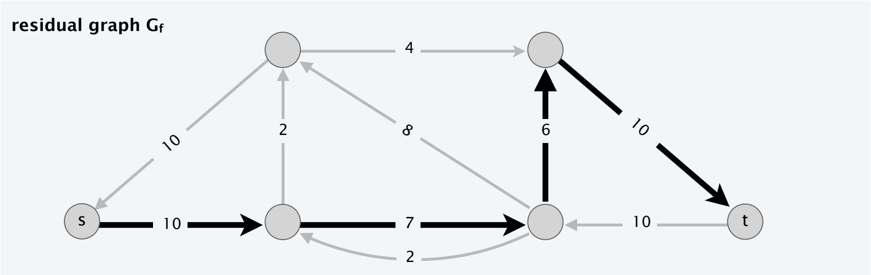
$bottleneck = 2$ $\Downarrow$



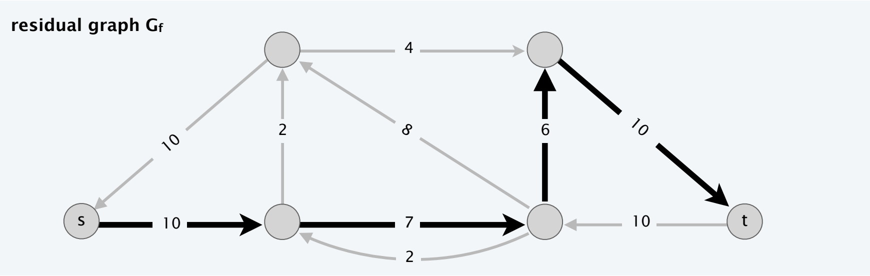
Ford-Fulkerson algorithm demo



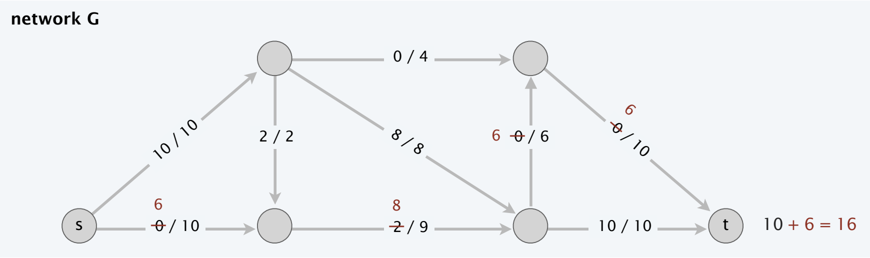
⇓



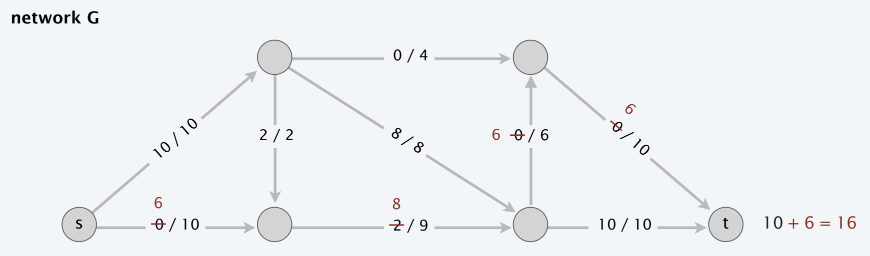
Ford-Fulkerson algorithm demo



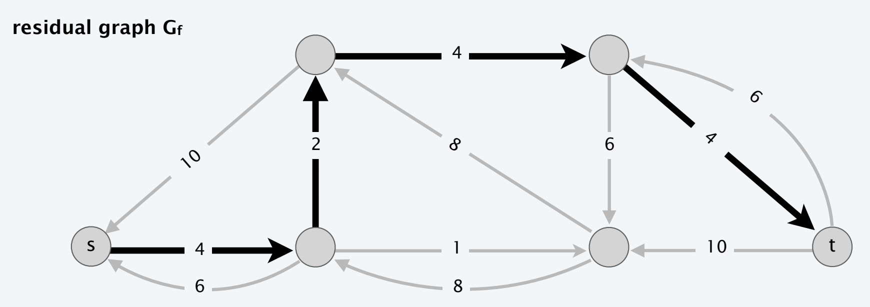
bottleneck $= 6$ $\Downarrow$

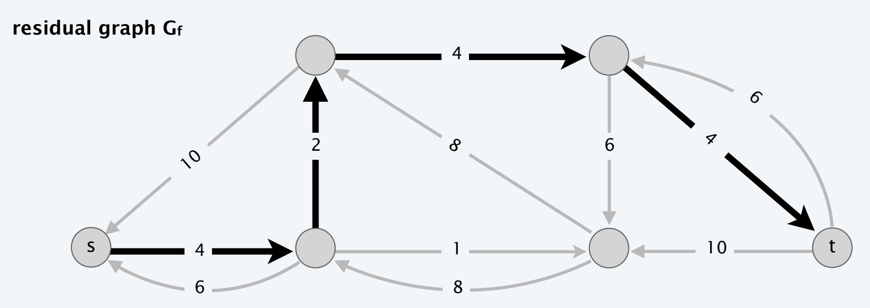


Ford-Fulkerson algorithm demo

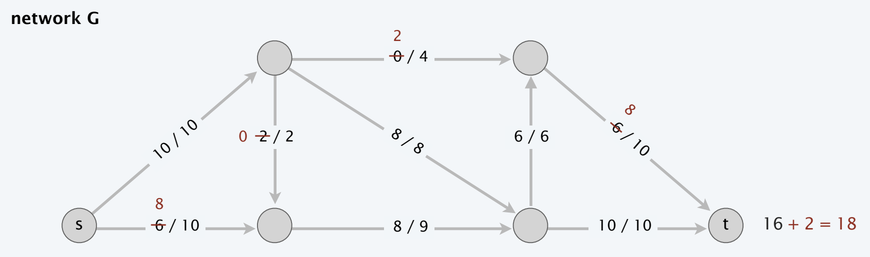


⇓

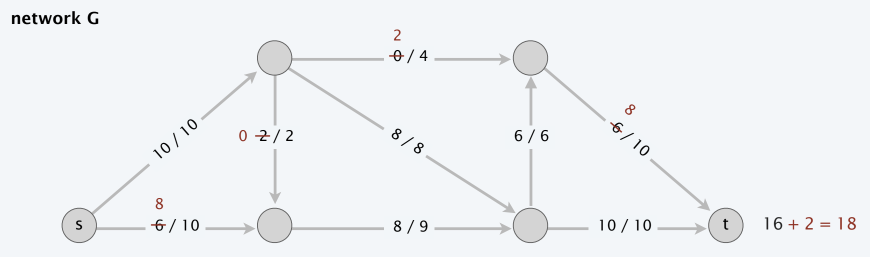




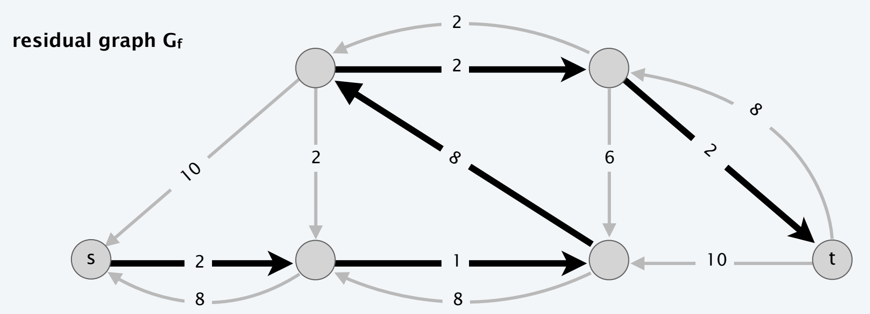
bottleneck $= 2$ $\Downarrow$



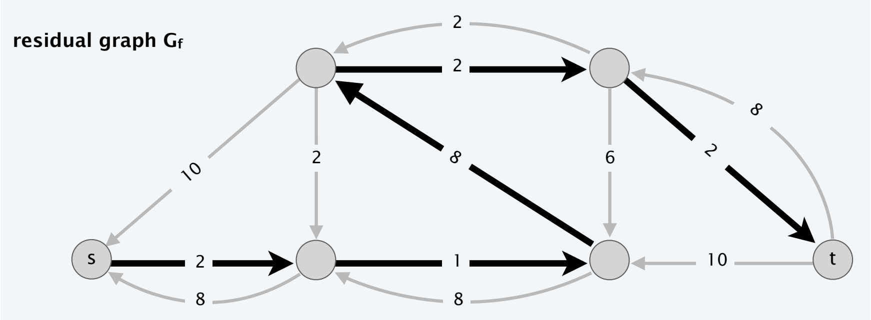
Ford-Fulkerson algorithm demo



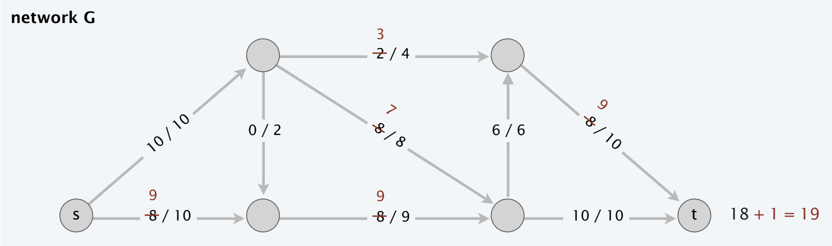
⇓

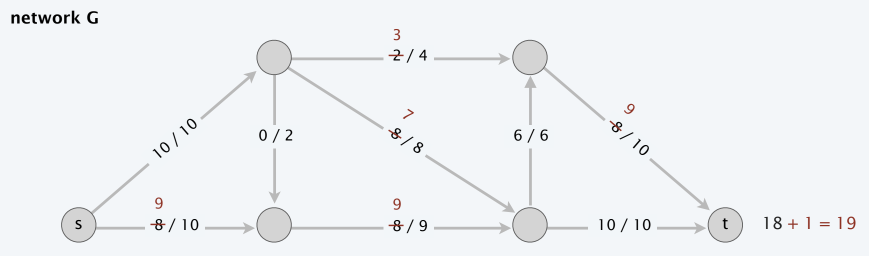


Ford-Fulkerson algorithm demo



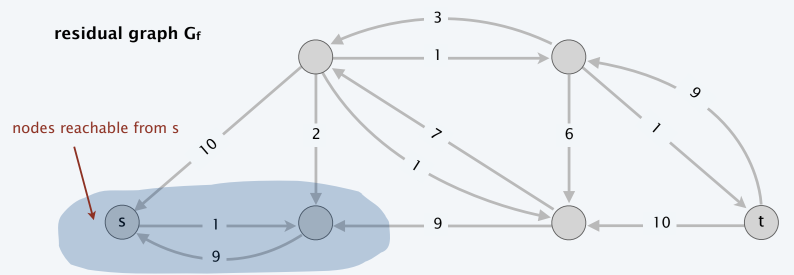
$bottleneck = 1$ $\Downarrow$



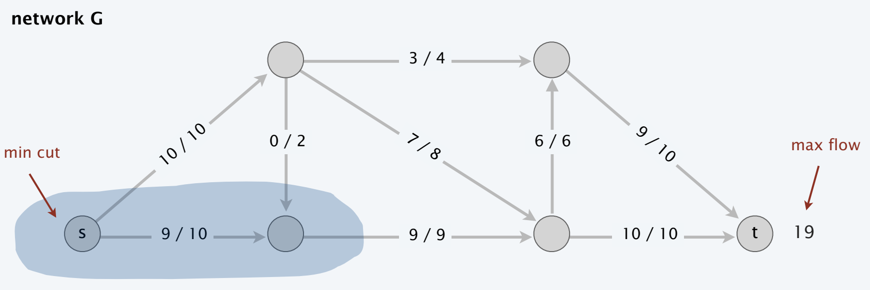


⇓

Residual graph G_f does not have any augmenting path!



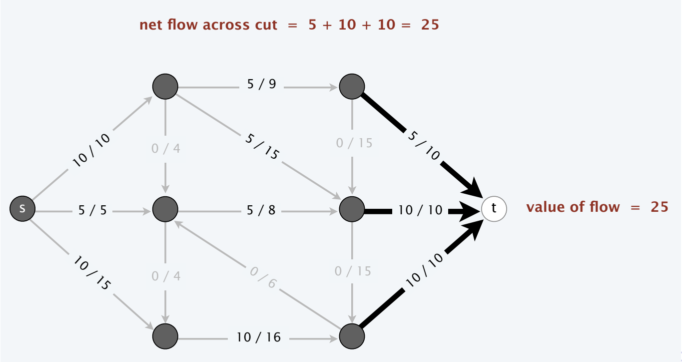
Ford-Fulkerson algorithm demo: Maximum Flow vs Minimum Cut



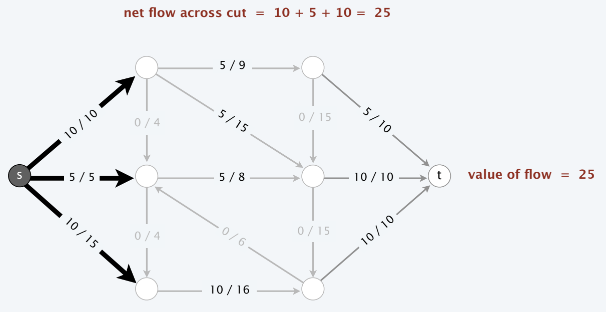
Lemma (Flow value)

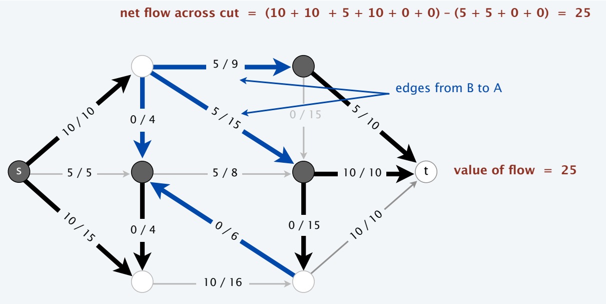
Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f:

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



Relationship between flows and cuts





Relationship between flows and cuts

Lemma (Flow value)

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

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

Proof.

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

{ by flow conservation, all terms except $v = s$ are 0, so}

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

□

Relationship between flows and cuts

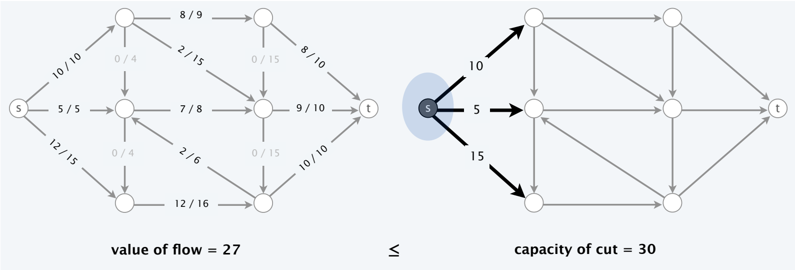
Fact (Weak duality)

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

Proof.

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \leq \sum_{e \text{ out of } A} f(e) \leq \sum_{e \text{ out of } A} c(e) = cap(A, B).$$

□



Max-flow min-cut theorem

Theorem (Max-flow min-cut theorem)

- *A flow f is a max-flow iff no augmenting paths.*
- *Value of the max-flow = capacity of min-cut.*

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.

$(1) \implies (2)$

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

($\leq$: weak duality; $=$: by assumption) □

Theorem (Max-flow min-cut theorem)

- *A flow f is a max-flow iff no augmenting paths.*
- *Value of the max-flow = capacity of min-cut.*

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) $\implies$ (3) We prove contrapositive: $\neg(3) \implies \neg(2)$.

- 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.

□

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.

$(3) \implies (1)$

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

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \sum_{e \text{ out of } A} c(e) = cap(A, B).$$

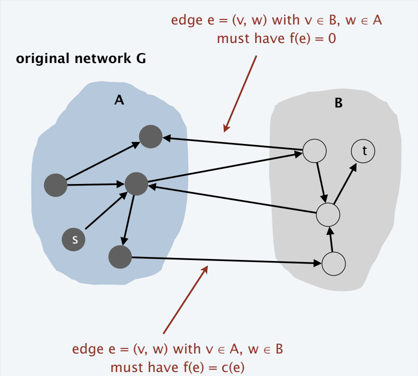
□

Proof.

(3) $\implies$ (1)

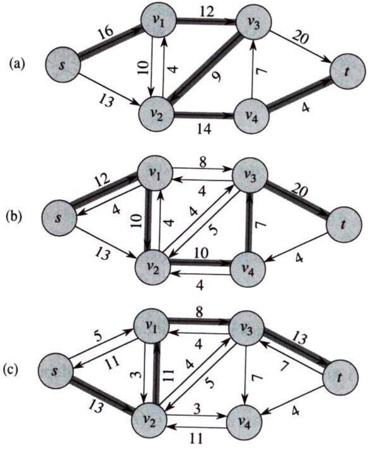
- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual graph G_f.
- By definition of cut A, $s \in A$.
- By definition of flow f, $t \notin A$.

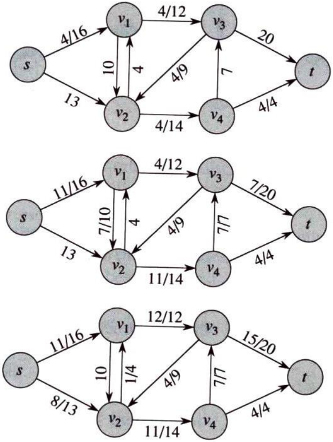
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \sum_{e \text{ out of } A} c(e) = cap(A, B).$$



G_f G

(a) if $f(e) = 0$ for all e the $G = G_f$



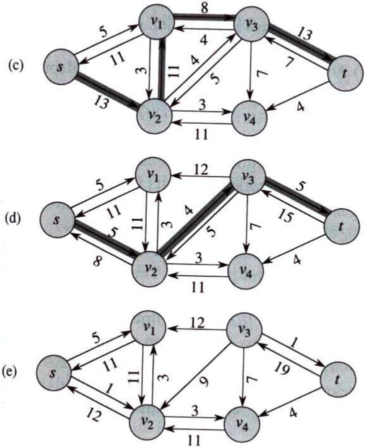


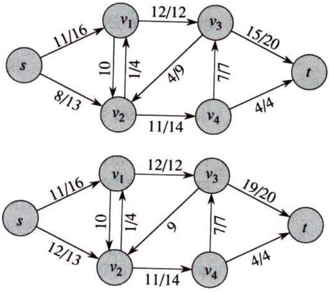
$val(f) = 4$

$val(f) = 11$

$val(f) = 19$

G_f G





$val(f) = 19$

$val(f) = 23$

$\Uparrow$

maximum flow $val(f^*) = 23$

no path from s to t

Running time of Ford-Fulkerson algorithm

FORD-FULKERSON (G, s, t, c)

FOREACH edge $e \in E : f(e) \leftarrow 0$.
  $G_f \leftarrow$ residual graph.
  WHILE (there exists an augmenting path P in G_f)
    $f \leftarrow$ AUGMENT (f, c, P).
    Update G_f.
  RETURN f.

Assumption. Capacities are positive integers.

Integrality invariant. Throughout the algorithm, the flow values $f(e)$ and the residual capacities $c_f(e)$ are integers.

Lemma (Maximal number of iterations)

The algorithm terminates in at most $val(f^*)$ iterations of the WHILE loop (f^* is the maximal flow).

Proof.

Each augmentation increases the value by at least 1. □

Running time of Ford-Fulkerson algorithm

FORD-FULKERSON (G, s, t, c)

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

$G_f \leftarrow$ residual graph.

WHILE (there exists an augmenting path P in G_f)

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

Update G_f.

RETURN f.

Assumption. Capacities are positive integers.

Integrality invariant. Throughout the algorithm, the flow values $f(e)$ and the residual capacities $c_f(e)$ are integers.

Theorem (Integrality theorem)

There exists a max-flow f^* for which every flow value $f^*(e)$ is an integer.

Proof.

Since algorithm terminates, theorem follows from invariant. □

Running time of Ford-Fulkerson algorithm

Theorem

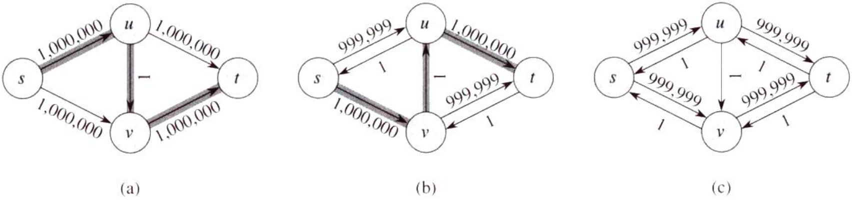
The total running time of Ford-Fulkerson algorithm is $O(m\ val(f^*))$, where m is the number of nodes.

Proof.

- By Lemma about maximal number of iterations, it suffice to show that what is inside of WHILE loop is $O(m)$.
- Update of G_f is clearly $O(m)$ as we just have to update each edge.
- The time to find a path in residual network is $O(m)$ is we use either depth-first search or breadth-first search (see Lecture Notes or other material for the second year algorithm course CS/SE 2C03).
- Hence what is inside of WHILE loop is $O(m)$, so the total time is $O(m\ val(f^*))$

□

Bad case of Ford-Fulkerson algorithm



An example of a flow network for which standard Ford-Fulkerson can take $\Theta(m\ val(f^*))$ time, where m is the number of edges and f^* is a maximum flow, which in this case is $2,000,000$.

Choosing good augmenting paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate! (see demo)

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting paths with:

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges: Edmonds-Karp Algorithm $O(nm^2)$, where n-number of vertices and m-number of edges.

Edmonds-Karp Algorithm

- When the augmenting path is always a **shortest** path from s to t in the residual graph G_f (assuming each edge has unit distance), and breadth-first search is used to find such a path, the algorithm is called *Edmonds-Karp Algorithm*, or *Shortest Augmenting Path Algorithm*.

SHORTEST-AUGMENTING-PATH(G, s, t, c)

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

$G_f \leftarrow$ residual graph.

WHILE (there exists an augmenting path in G_f)

$P \leftarrow$ BREADTH-FIRST-SEARCH (G_f, s, t).

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

Update G_f.

RETURN f.

- Time complexity of Edmonds-Karp Algorithm is $O(m^2 n)$.
- There are algorithms with complexities $O(n^2 m)$ and $O(n^3)$.