Algorithm Theory

Chapter 6
Graph Algorithms

Maximum Flow: Ford Fulkerson Algorithm

Fabian Kuhn

Graphs

Extremely important concept in computer science

Graph $\boldsymbol{G = (V, E)}$

- V: node (or vertex) set
- $E \subseteq V^2$: edge set
  - undirected graph: we often think of edges as sets of size 2 (e.g., $\{u, v\}$)
  - directed graph (digraph): edges are sometimes also called arcs
  - simple graph: no self-loops, no multiple edges
  - weighted graph: (positive) weight on edges (or nodes)
- (simple) path: sequence $v_0, \dots, v_k$ of nodes such that
  $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \dots, k-1\}$
- …

Many real-world problems can be formulated as optimization problems on graphs.

Graph Optimization: Examples

Minimum spanning tree (MST):

- Compute min. weight spanning tree of a weighted undir. Graph

Shortest paths:

- Compute (length) of shortest paths (single source, all pairs, …)

Traveling salesperson (TSP):

- Compute shortest TSP path/tour in weighted graph

Vertex coloring:

- Color the nodes such that neighbors get different colors
- Goal: minimize the number of colors

Maximum matching:

- Matching: set of pair-wise non-adjacent edges
- Goal: maximize the size of the matching

Network Flow

Flow Network:

- Directed graph $G = (V, E), E \subseteq V^2$
- Each (directed) edge e has a capacity $c_e \geq 0$
  - Amount of flow (traffic) that the edge can carry
- A single source node $s \in V$ and a single sink node $t \in V$
  - Source s has only outgoing edges, sink t has only incoming edges

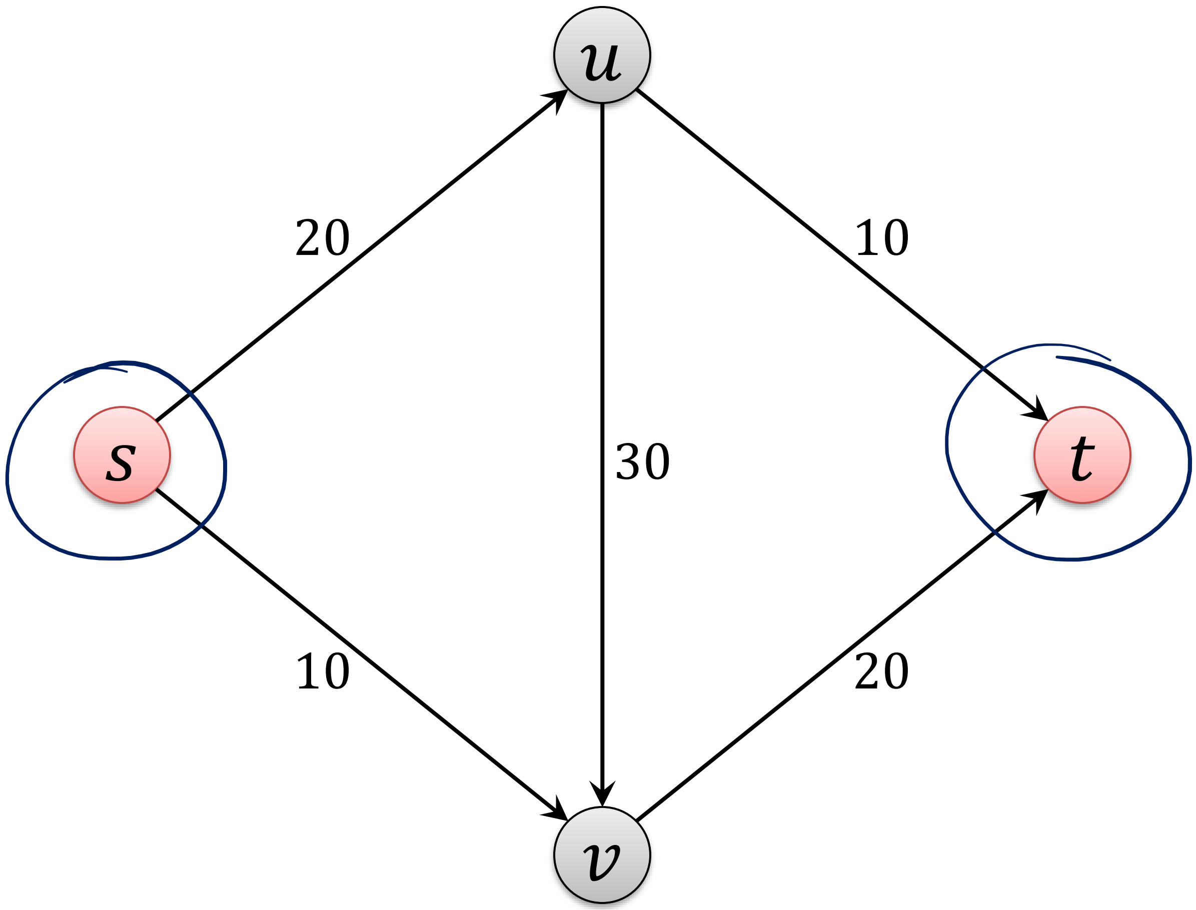
Flow: (informally)

- Traffic from s to t such that each edge carries at most its capacity

Examples:

- Highway system: edges are highways, flow is the traffic
- Computer network: edges are network links, flow is data
- Fluid network: edges are pipes that carry liquid

Example: Flow Network



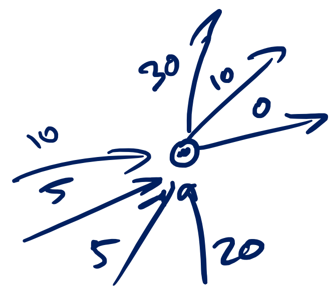
Network Flow: Definition

Flow: function $\boldsymbol{f}: \boldsymbol{E} \to \mathbb{R}_{\geq 0}$ $\quad f(e) \geq 0$ $\quad$ cap $\overset{10}{e}$

- $f(e)$ is the amount of flow carried by edge e $\quad 0 \leq f(e) \leq 10$

Capacity Constraints:

- For each edge $e \in E$, $f(e) \leq c_e$



Flow Conservation:

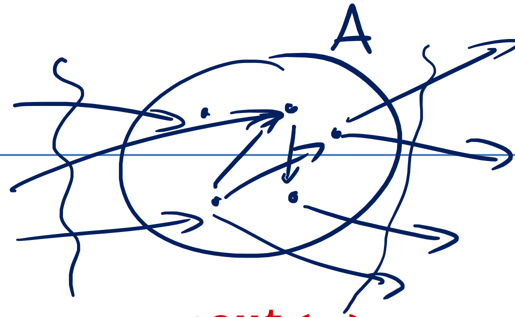
- For each node $v \in V \setminus \{s, t\}$,

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

Flow Value:

$$|f| := \sum_{e \text{ out of } s} f\big((s,u)\big) = \sum_{e \text{ into } t} f\big((v,t)\big)$$

Notation



We define:

$$f^{\text{in}}(v) := \sum_{e \text{ into } v} f(e), \qquad f^{\text{out}}(v) := \sum_{e \text{ out of } v} f(e)$$

For a set $A \subseteq V$:

$$f^{\text{in}}(A) := \sum_{e \text{ into } A} f(e), \qquad f^{\text{out}}(A) := \sum_{e \text{ out of } S} f(e)$$

Flow conservation: $\forall v \in V \setminus \{s, t\}$: $f^{\text{in}}(v) = f^{\text{out}}(v)$

Flow value: $|f| = f^{\text{out}}(s) = f^{\text{in}}(t)$

For simplicity: Assume that all capacities are positive integers

The Maximum-Flow Problem

Maximum Flow:

Given a flow network, find a flow of maximum possible value

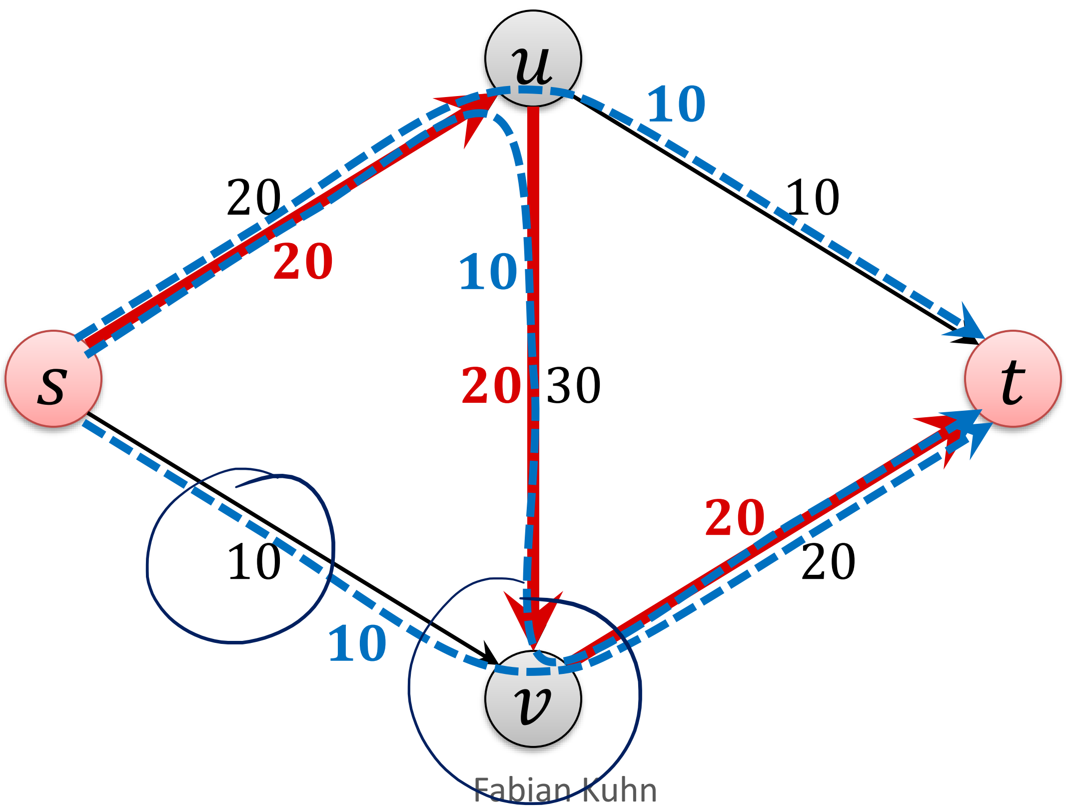
- Classic graph optimization problem
- Many applications (also beyond the obvious ones)
- Requires new algorithmic techniques

Maximum Flow: Greedy?

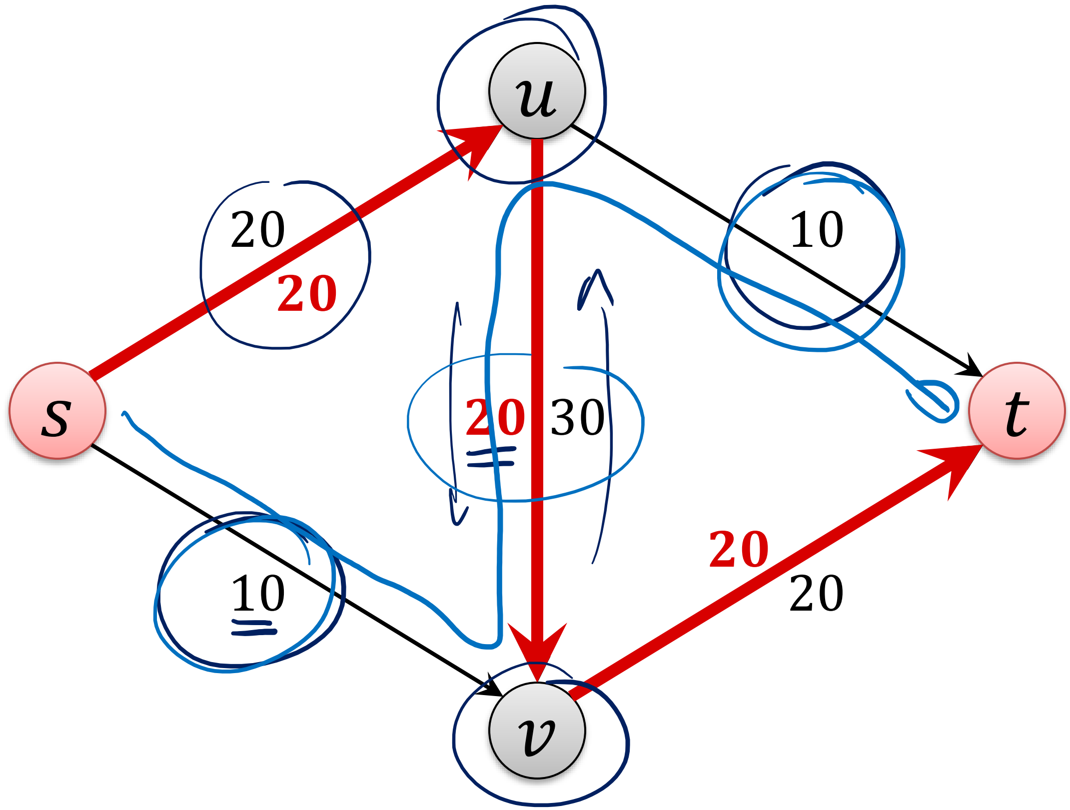
Does greedy work?

A natural greedy algorithm:

- As long as possible, find an s-t-path with free capacity and add as much flow as possible to the path



Improving the Greedy Solution



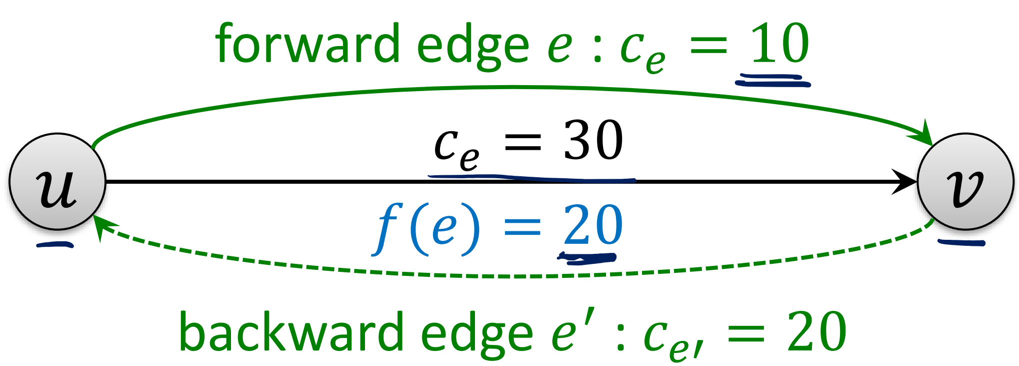
- Try to push 10 units of flow on edge (s, v)
- Too much incoming flow at v: reduce flow on edge (u, v)
- Add that flow on edge (u, t)

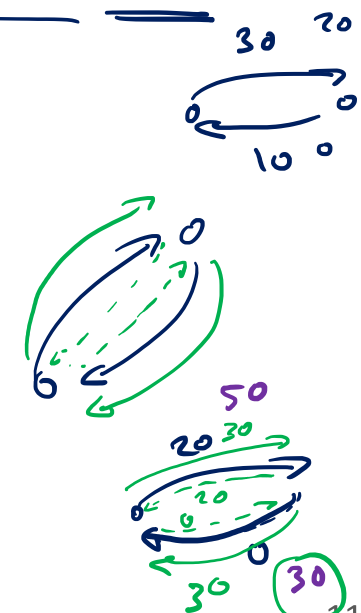
Residual Graph

Given a flow network $G = (V, E)$ with capacities c_e (for $e \in E$)

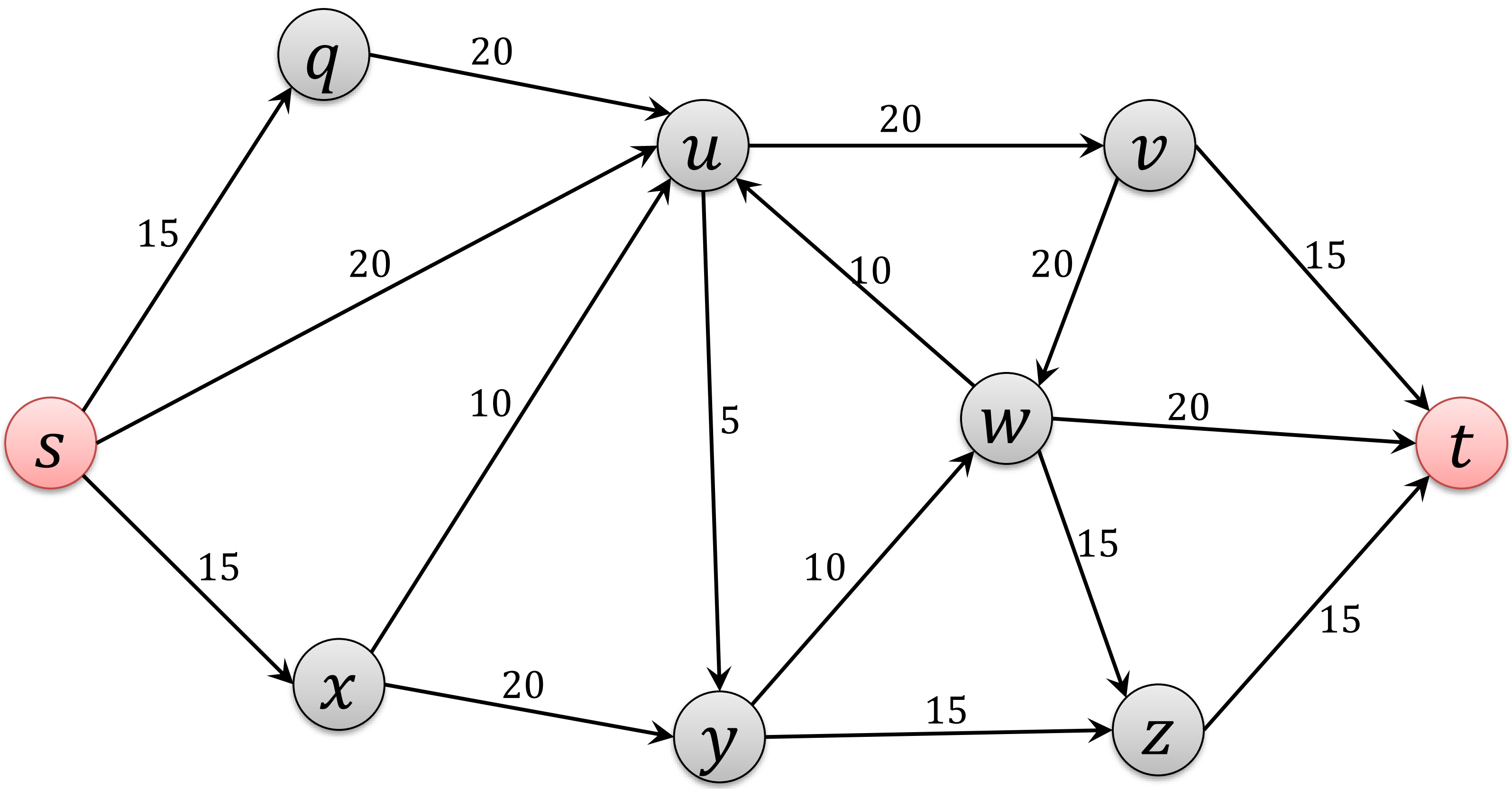
For a flow f on G, define directed graph $G_f = (V_f, E_f)$ as follows:

- Node set $V_f = V$
- For each edge $e = (u, v)$ in E, there are two edges in E_f:
  - forward edge $e = (u, v)$ with residual capacity $c_e - f(e)$
  - backward edge $e' = (v, u)$ with residual capacity $f(e)$



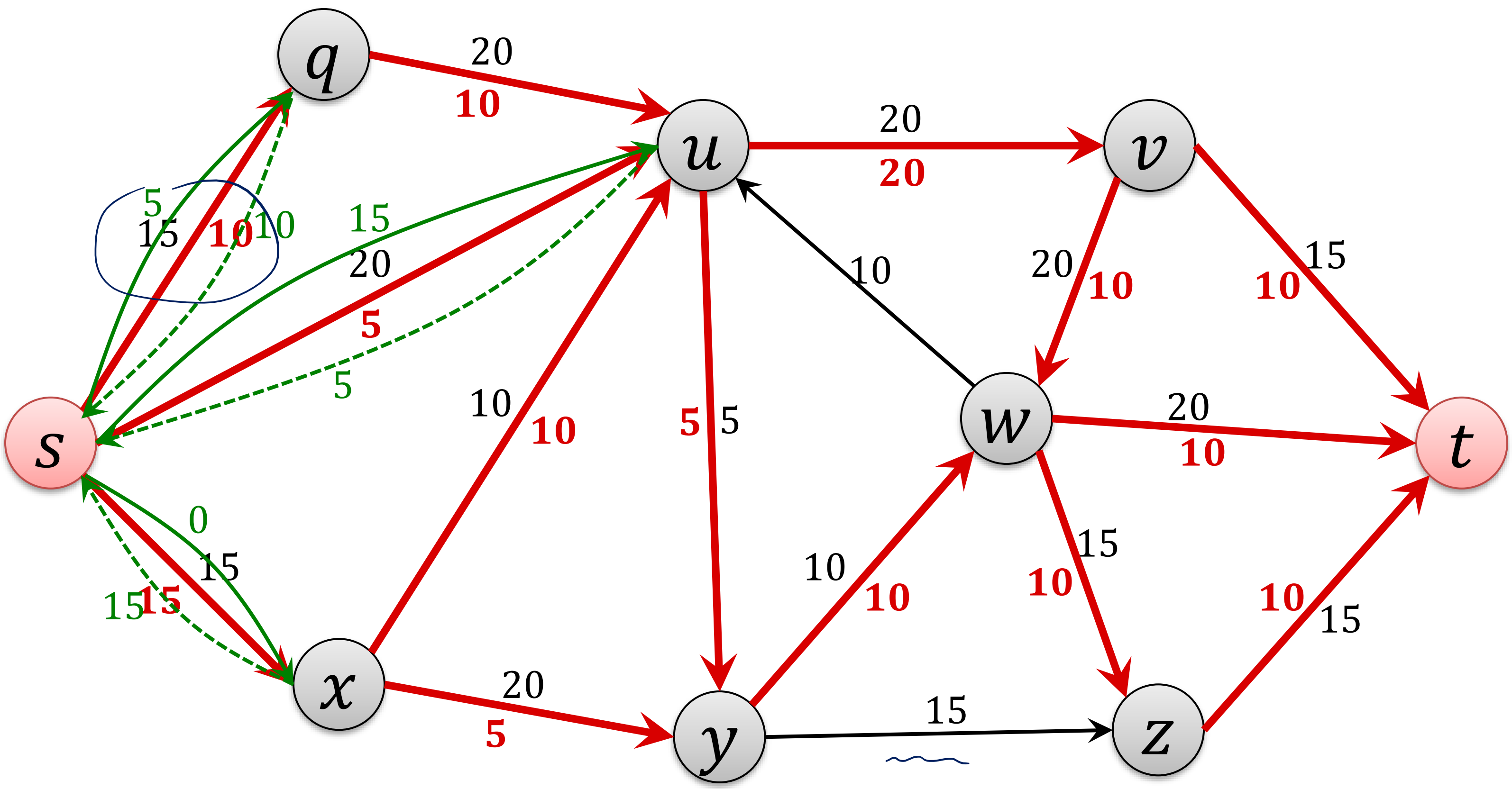


Residual Graph: Example



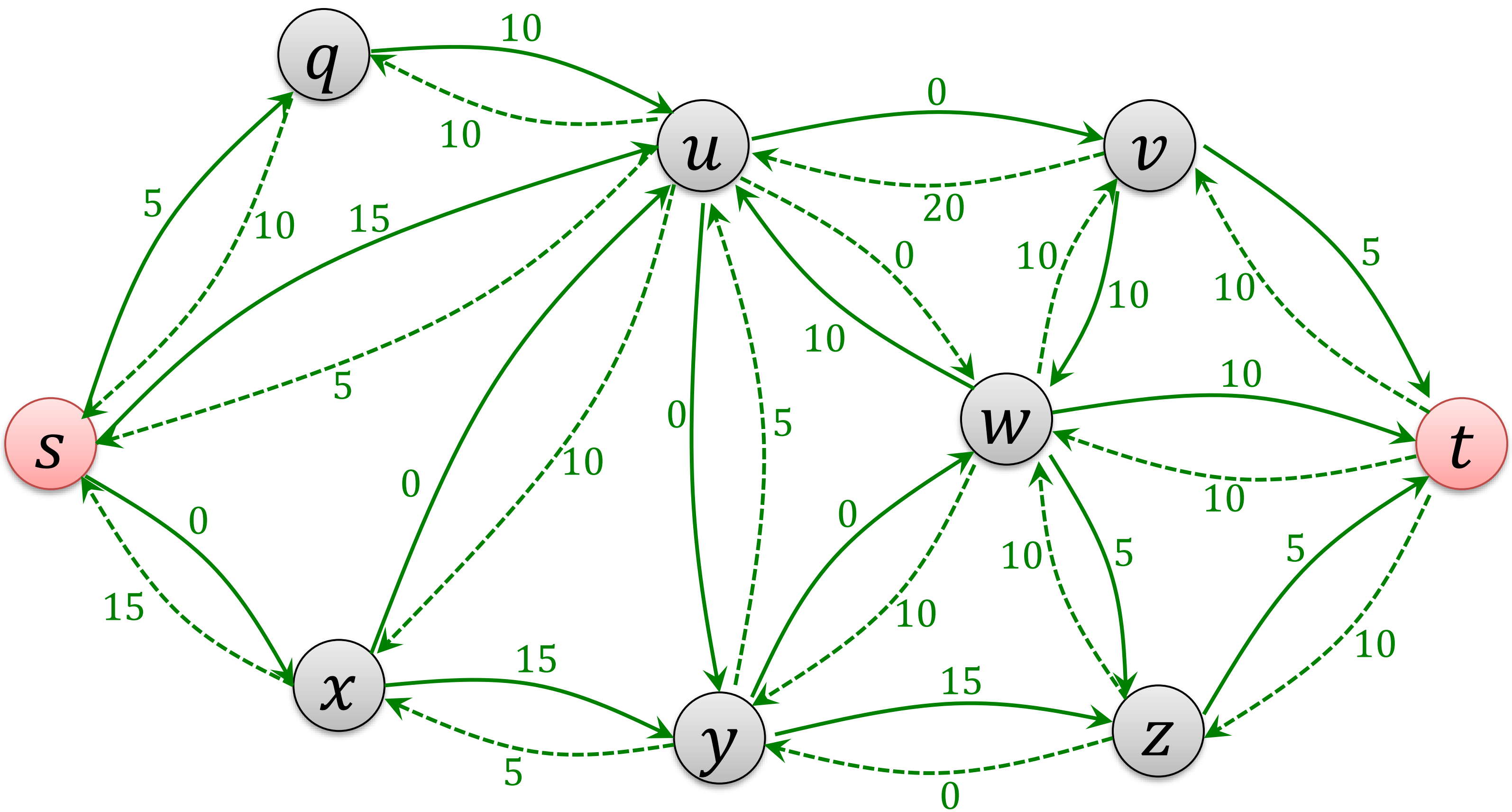
Residual Graph: Example

Flow f



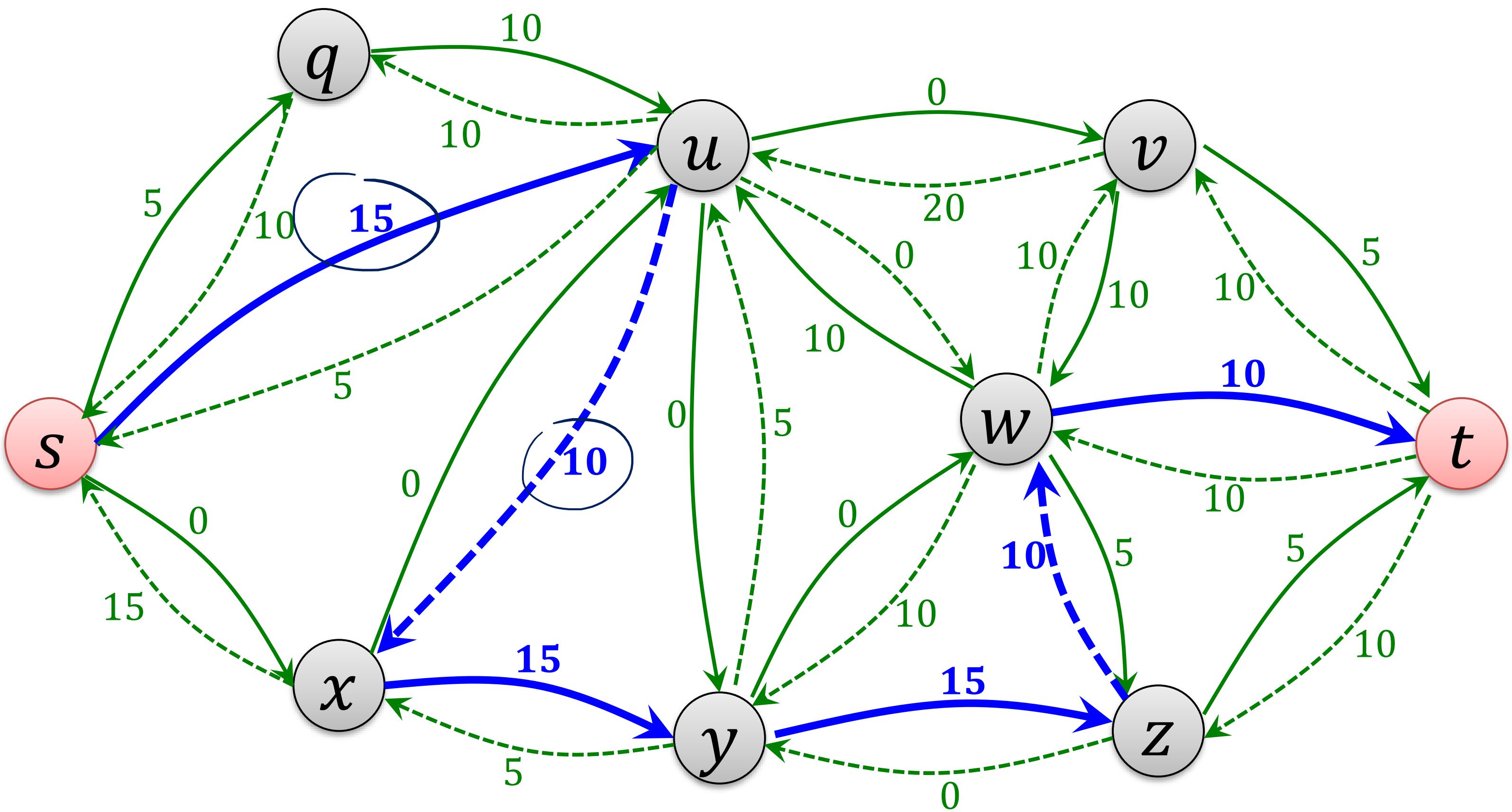
Residual Graph: Example

Residual Graph G_f



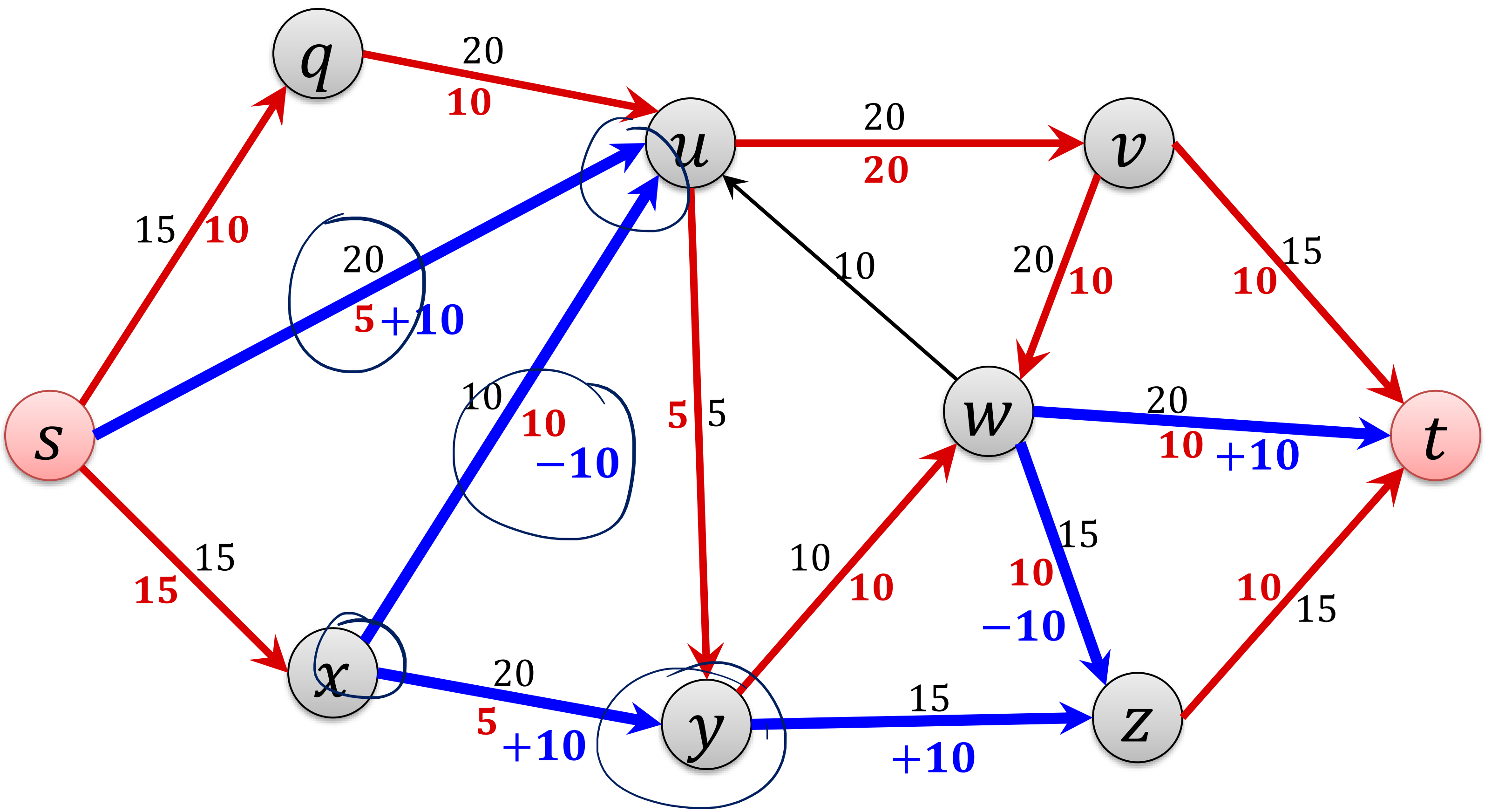
Augmenting Path

Residual Graph G_f



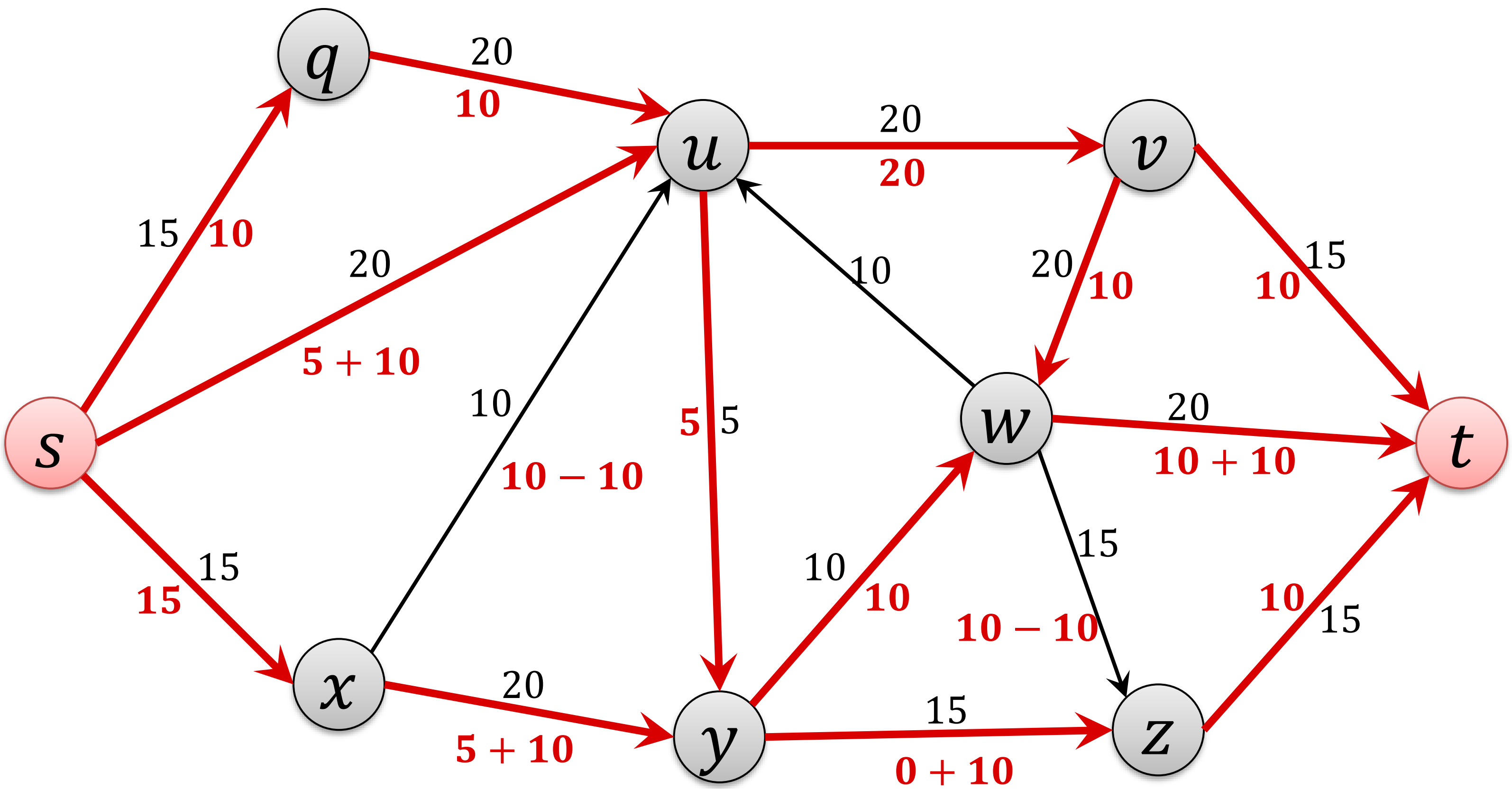
Augmenting Path

Augmenting Path



Augmenting Path

New Flow



Augmenting Path

Definition:
An augmenting path P is a (simple) s-t-path on the residual graph G_f on which each edge has residual capacity > 0.

$\text{bottleneck}(P, f)$: minimum residual capacity on any edge of the augmenting path P

Augment flow f to get flow f':

- For every forward edge (u, v) on P:

$$f'\big((u,v)\big) \coloneqq f\big((u,v)\big) + \textbf{bottleneck}(P, f)$$

- For every backward edge (u, v) on P:

$$f'\big((v,u)\big) \coloneqq f\big((v,u)\big) - \textbf{bottleneck}(P, f)$$

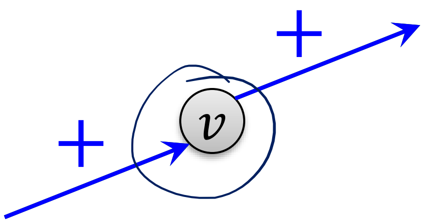
Augmented Flow

Lemma: Given a flow f and an augmenting path P, the resulting augmented flow f' is legal and its value is

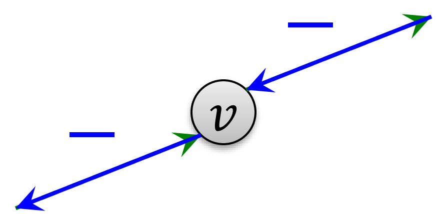
$$|f'| = |f| + \textbf{bottleneck}(P, f).$$

Proof:

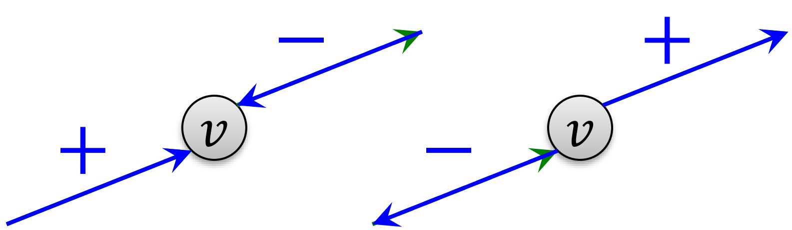
2 forward edges



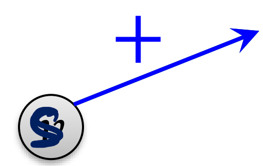
2 backward edges



forward & backward edge



flow value increases



Ford-Fulkerson Algorithm

- Improve flow using an augmenting path as long as possible:

1. Initially, $f(e) = 0$ for all edges $e \in E$, $G_f = G$
2. **while** there is an augmenting s-t-path P in G_f **do**
3. Let P be an augmenting s-t-path in G_f;
4. $f' \coloneqq \text{augment}(f, P)$;
5. update f to be f';
6. update the residual graph G_f
7. **end**;

Ford-Fulkerson Running Time

Theorem: If all edge *capacities* are *integers*, the Ford-Fulkerson algorithm terminates after at most C iterations, where

$$C = \text{"max flow value"} \leq \sum_{e \text{ out of } s} c_e .$$

Proof:

1. At all times, for all $e \in E$, $f(e)$ is an integer
   - Initially: $f(e) = 0$
   - In one iteration:
     - augmenting path P: all residual capacities are integers
     - $\text{bottleneck}(P, f) > 0$ and also $\text{bottleneck}(P, f)$ is an integer
     - $f'(e) = f(e)$ or $f'(e) = f(e) \pm \text{bottleneck}(P, f)$

2. New flow value $|f'| = |f| + \text{bottleneck}(P, f) \geq |f| + 1$

   $\Longrightarrow$ **#iterations $\leq C$**

Ford-Fulkerson Running Time

Theorem: If all edge *capacities* are *integers*, the Ford-Fulkerson algorithm can be implemented to run in $O(mC)$ time.

m: #edges

Proof:

Show that each of the $\leq C$ iterations requires $O(m)$ time.

1. Compute / update residual graph:
   - 1st iteration: $O(m)$
   - Later iterations: $O(n)$
2. Find augmenting path / conclude that no augm. path exists

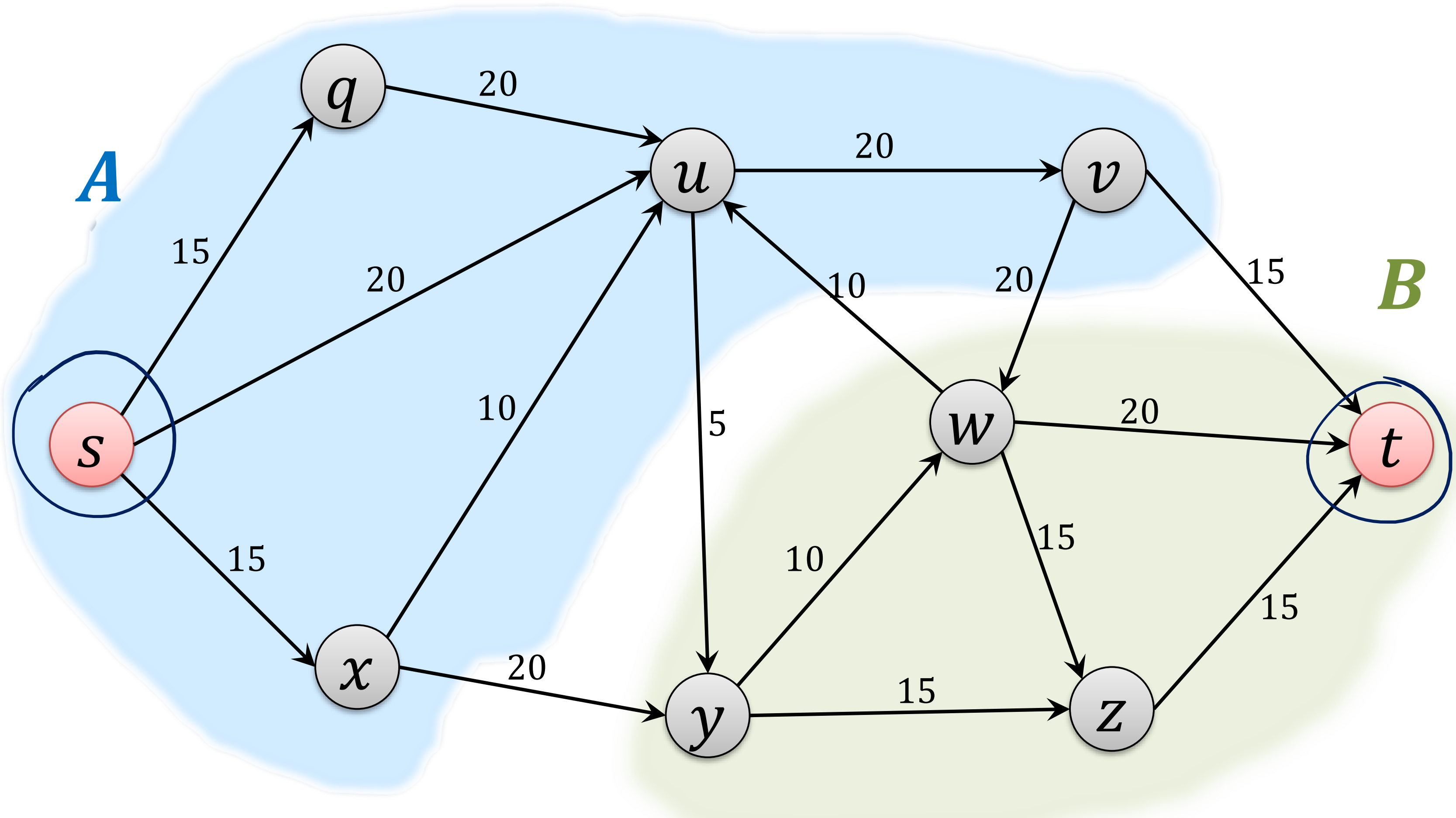
   find positive s-t path in residual graph G_f

   $\Longrightarrow$ Graph traversal: using DFS or BFS: $O(m)$

   $O(m+n)$
3. Update flow values: $O(n)$

s-t Cuts

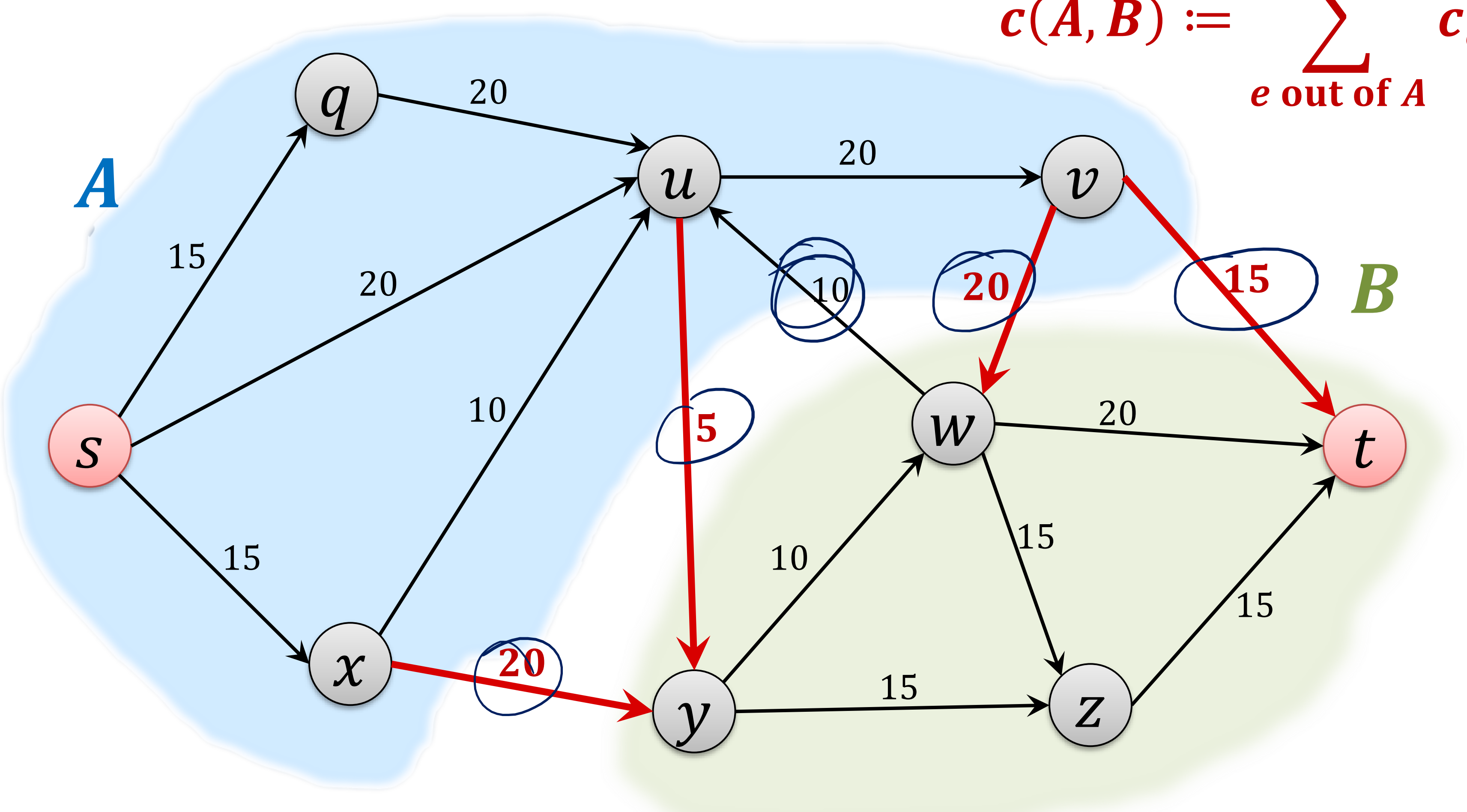
Definition:
An s-t cut is a partition (A, B) of the vertex set such that $s \in A$ and $t \in B$



Cut Capacity

Definition:
The capacity $c(A, B)$ of an s-t-cut (A, B) is defined as

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



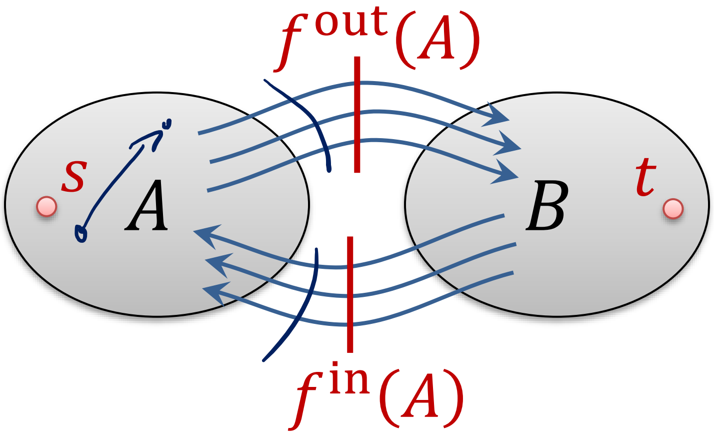
Cuts and Flow Value

Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|\boldsymbol{f}| = \boldsymbol{f}^{\mathbf{out}}(\boldsymbol{A}) - \boldsymbol{f}^{\mathbf{in}}(\boldsymbol{A}).$$

Proof:

$$|f| = f^{\text{out}}(s), \qquad \left(= f^{\text{in}}(t)\right)$$



$$|f| = f^{\text{out}}(s) - \underbrace{f^{\text{in}}(s)}_{=0}$$

$$= \sum_{v \in A} \Big(\underbrace{f^{\text{out}}(v) - f^{\text{in}}(v)}_{=\,0,\ \text{except for}\ v\,=\,s} \Big)$$

$$= f^{\text{out}}(A) - f^{\text{in}}(A)$$

Cuts and Flow Value

Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|\boldsymbol{f}| = \boldsymbol{f}^{\text{out}}(\boldsymbol{A}) - \boldsymbol{f}^{\text{in}}(\boldsymbol{A}).$$

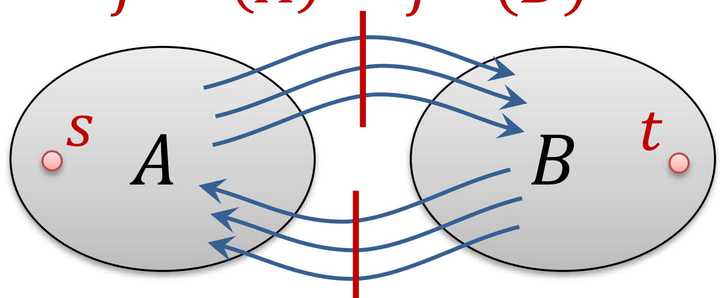
Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|\boldsymbol{f}| = \boldsymbol{f}^{\text{in}}(\boldsymbol{B}) - \boldsymbol{f}^{\text{out}}(\boldsymbol{B}).$$

Proof:

- Either do the same argument as before, symmetrically
- Or, use that $f^{\text{out}}(A) = f^{\text{in}}(B)$ and $f^{\text{in}}(A) = f^{\text{out}}(B)$

$$f^{\text{out}}(A) = f^{\text{in}}(B)$$



$$f^{\text{in}}(A) = f^{\text{out}}(B)$$

Upper Bound on Flow Value

Lemma:

Let f be any s-t flow and (A, B) any s-t cut. Then $|\boldsymbol{f}| \leq \boldsymbol{c(A, B)}$.

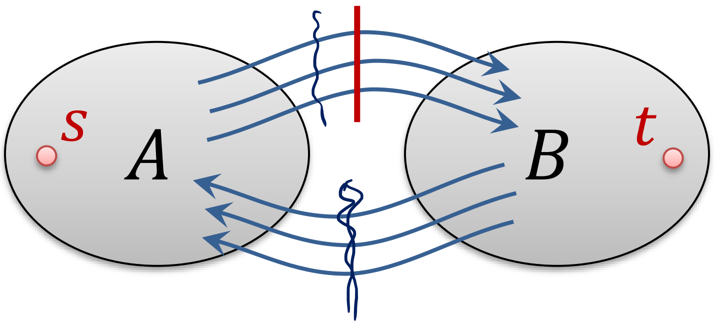
Proof:

$$\underline{|f|} = \underline{f^{\text{out}}}(A) - \underline{f^{\text{in}}}(A) \leq c(A, B)$$

$$\underline{f^{\text{out}}(A) \leq c(A, B)}$$

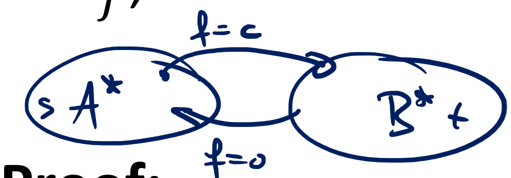
$$f^{\text{in}}(A) \underline{\geq 0}$$

$= c(A, B)$



Ford-Fulkerson Gives Optimal Solution

Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

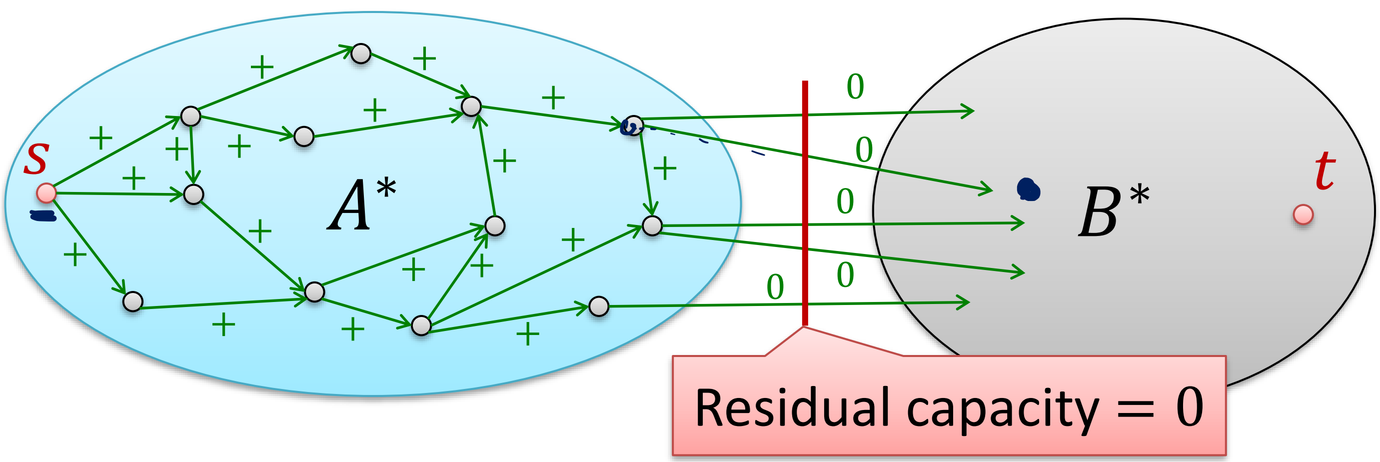


$$|\boldsymbol{f}| = \boldsymbol{c}(\boldsymbol{A}^*, \boldsymbol{B}^*).$$

$|f^*| \leq c(A^*, B^*)$

Proof:

- Define $\boldsymbol{A}^*$: set of nodes that can be reached from s on a path with positive residual capacities in G_f:



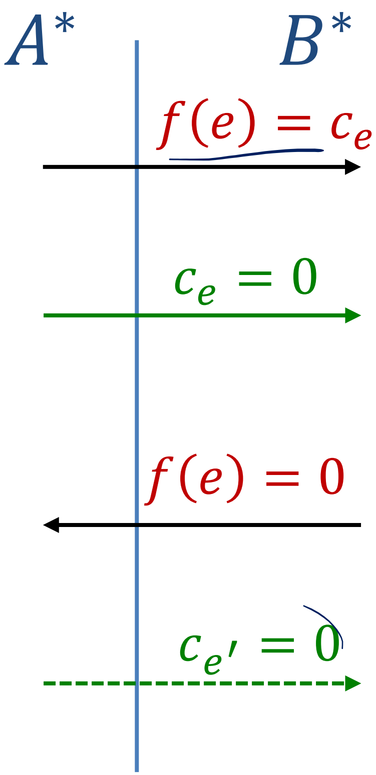
- For $B^* = V \setminus A^*$, (A^*, B^*) is an s-t cut
  - By definition $s \in A^*$ and $t \notin A^*$

Ford-Fulkerson Gives Optimal Solution

Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

$$|\boldsymbol{f}| = \boldsymbol{c}(\boldsymbol{A}^*, \boldsymbol{B}^*).$$

Proof:



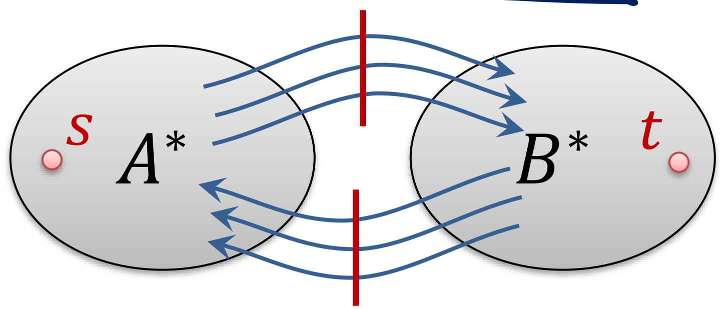
Edge e from A^* to B^* in G

Forward edge e in G_f

Edge e from B^* to A^* in G

Backward edge e' in G_f

$$f^{\text{out}}(A^*) = c(A^*, B^*)$$



$$f^{\text{in}}(A^*) = 0$$

Ford-Fulkerson Gives Optimal Solution

Theorem: The flow returned by the Ford-Fulkerson algorithm is a maximum flow.

Proof:

- Ford-Fulkerson algorithm gives a flow f^* and a cut (A^*, B^*)

$$\text{s.t.} \quad |f^*| = c(A^*, B^*).$$

- We saw that $|f| \leq c(A, B)$ for every valid flow f and every s-t cut (A, B).
  - And thus in particular also $|f| \leq c(A^*, B^*)$.

Min-Cut Algorithm

Ford-Fulkerson also gives a minimum s-t cut algorithm:

Theorem: Given a flow f of maximum value, we can compute an s-t cut of minimum capacity in $O(m)$ time.

Proof:

- f maximum $\Longrightarrow$ no augmenting path
- We can therefore construct cut (A^*, B^*) as before
  - By using DFS/BFS on the positive res. cap. edges of G_f in time $O(m)$.

- (A^*, B^*) is a cut of minimum capacity:
  - For every other s-t cut (A, B), we know that $|f| \leq c(A, B)$
  - Because $|f| = c(A^*, B^*)$, we therefore have

$$c(A^*, B^*) \leq c(A, B).$$

Max-Flow Min-Cut Theorem

Theorem: (Max-Flow Min-Cut Theorem)

In every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.

Proof:

- Ford-Fulkerson gives a maximum flow f^* and a minimum cut (A^*, B^*) s.t.

$$|f^*| = c(A^*, B^*).$$

Integer Capacities

Theorem: **(Integer-Valued Flows)**

If all capacities in the flow network are integers, then there is a maximum flow f for which the flow $f(e)$ of every edge e is an integer.

Proof:

- If all the capacities are integers, the Ford-Fulkerson algorithm gives an integer solution.
  - By induction on the steps of the algorithm, all flow values are always integers and all residual capacities of G_f are always integers.

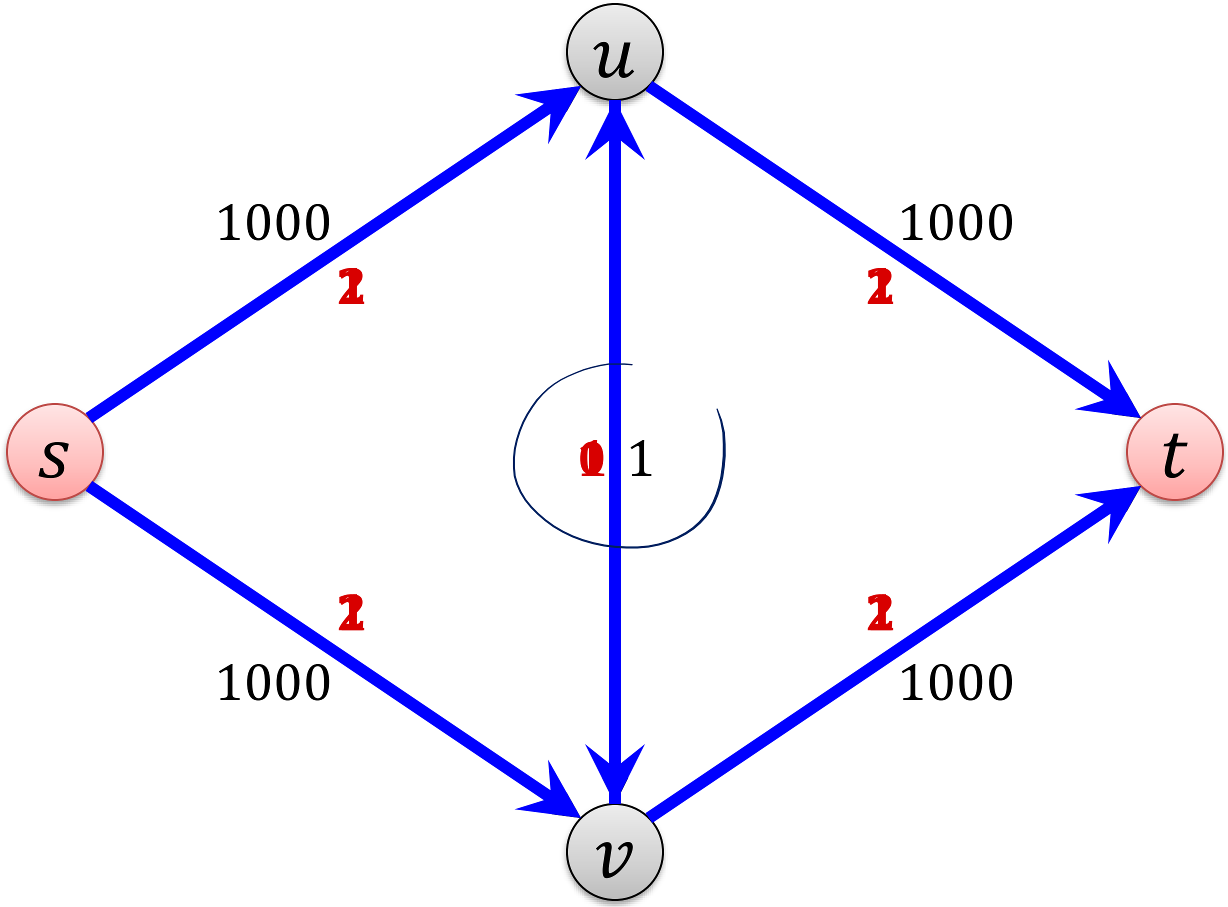
Non-Integer Capacities

If a given flow network has integer capacities, the Ford-Fulkerson algorithm computes a maximum flow of value C in time $O(m \cdot C)$.

What if capacities are not integers?

- rational capacities:
  - can be turned into integers by multiplying them with large enough integer
  - algorithm still works correctly
- real (non-rational) capacities:
  - not clear whether the algorithm always terminates
- even for integer capacities, time can linearly depend on the value of the maximum flow

Slow Execution



- Number of iterations: 2000 (value of max. flow)

Improved Algorithm

Idea: Find the best augmenting path in each step

- best: path P with maximum $\text{bottleneck}(P, f)$

- Best path might be rather expensive to find

  → find almost best path

$\max_{e \in E} c_e$

- **Scaling parameter Δ:**
  (initially, $\Delta =$ "$\max c_e$ rounded down to next power of 2")

- As long as there is an augmenting path that improves the flow by at least Δ, augment using such a path

- If there is no such path: $\Delta := \Delta/_2$

Scaling Parameter Analysis

$2^{\lfloor \log_2 c_{max} \rfloor}$

Lemma: If all capacities are integers, number of different scaling parameters used is $\leq 1 + \lfloor \log_2 c_{\max} \rfloor$.

$c_{\max} := \max_e c_e$

At the beginning: $\Delta = 2^{\lfloor \log_2 c_{\max} \rfloor}$

At the end: $\Delta = 1$

different scaling parameters Δ: $\lfloor \log_2 c_{\max} \rfloor + 1$

- **Δ-scaling phase:** Time during which scaling parameter is Δ

$$\text{running time} = \underbrace{\text{\#scaling phases}}_{O(\log c_{\max})} \cdot \underbrace{\text{\#iterations per phase}}_{???} \cdot O(m)$$

Length of a Scaling Phase

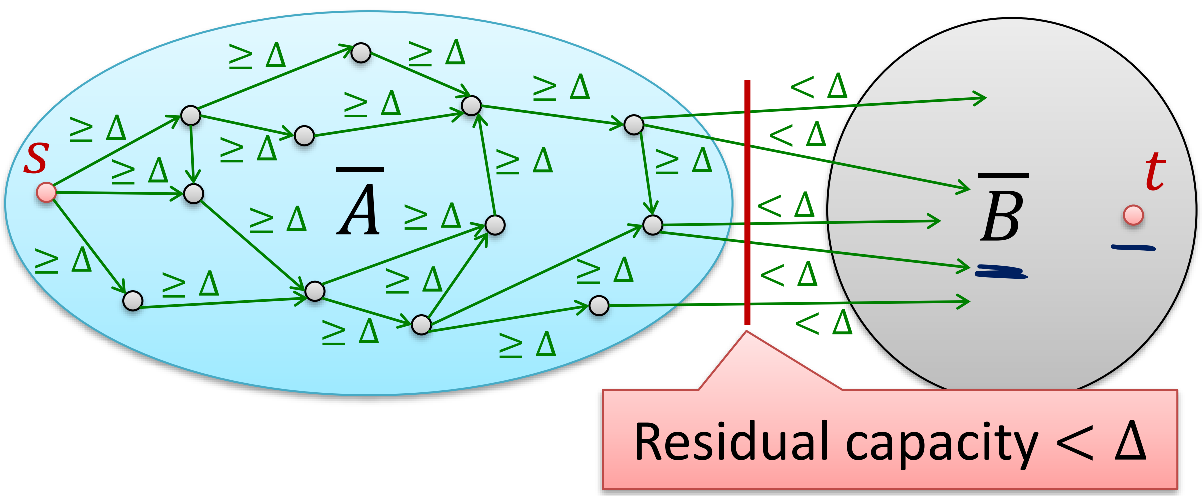
20

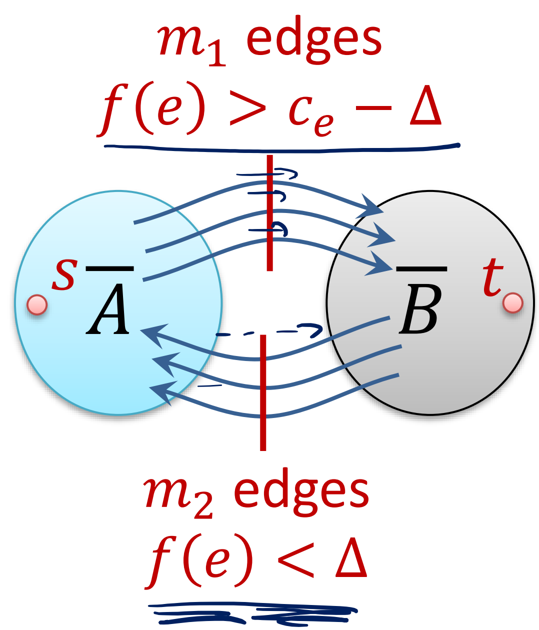
1.5Δ

Lemma: If f is the flow at the end of the Δ-scaling phase, the maximum flow in the network has value less than $|f| + m\Delta$.

Proof:

- Define $\overline{A}$: set of nodes that can be reached from s on a path with residual capacities $\geq \Delta$ in G_f.





$$|f| = f^{\text{out}}(\overline{A}) - f^{\text{in}}(\overline{A}) > c(\overline{A}, \overline{B}) - m_1\Delta - m_2\Delta \geq c(\overline{A}, \overline{B}) - m\Delta$$

Length of a Scaling Phase

Lemma: The number of augmentation in each scaling phase is less than $2m$.

Proof:

- At the end of the 2Δ-scaling phase: $|f^*| < |f| + 2m\Delta$
- Each augmentation in the Δ-scaling phase improves the value of the flow f by at least Δ.
- #augmentations in Δ-scaling phase $< 2m$.

Running Time: Scaling Max Flow Alg.

Theorem: The number of augmentations of the algorithm with scaling parameter and integer capacities is at most $O(m \log c_{\max})$. The algorithm can be implemented in time $O(m^2 \log c_{\max})$.

Proof:

- #scaling phases: $O(\log c_{\max})$
- #iterations per scaling phase: $O(m)$
- time per iteration: $O(m)$

Strongly Polynomial Algorithm

- Time of regular Ford-Fulkerson algorithm with integer capacities:

$$O(mC)$$

- Time of algorithm with scaling parameter:

$$O(m^2 \log c_{\max})$$

#edges

#nodes

- $O(\log c_{\max})$ is polynomial in the size of the input, but not in n
- Can we get an algorithm that runs in time polynomial in n?
- **Edmonds-Karp Alg.:** Always picking a shortest augmenting path:

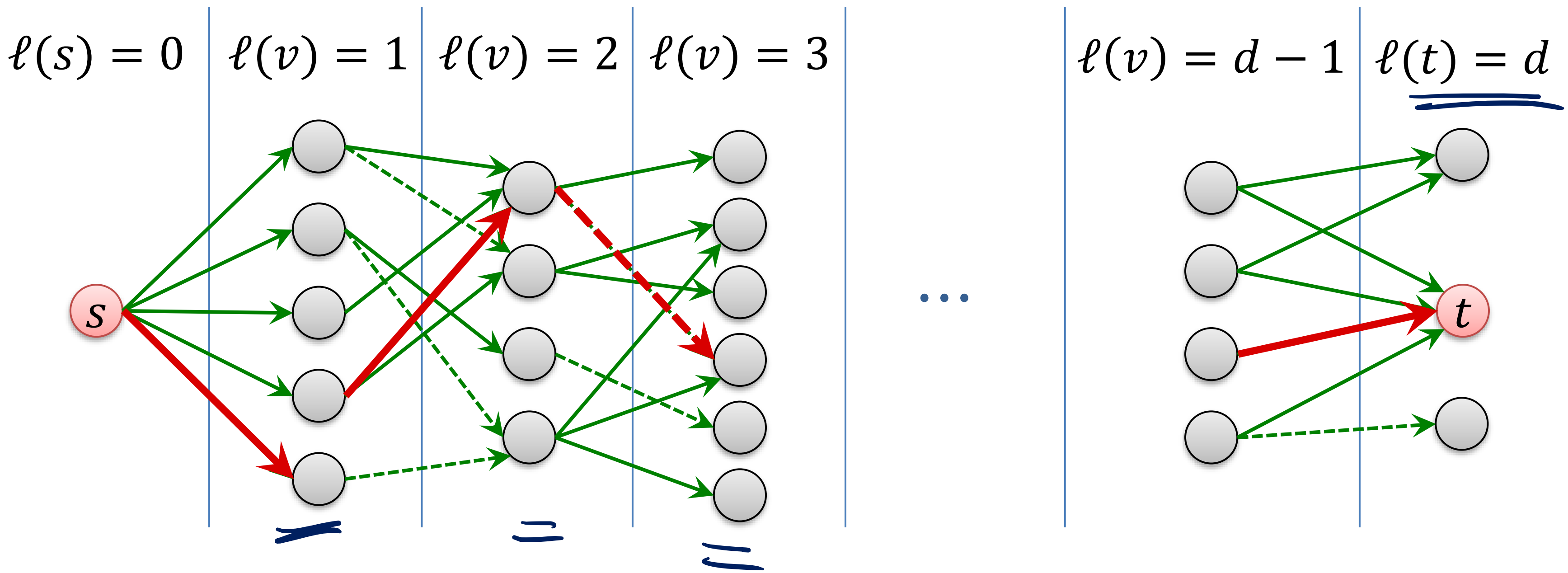
$$O(m^2 n)$$

  – also works for arbitrary real-valued weights
  – We will show this next.

Shortest Augmenting Path Algorithm

- Define G_f^+ as the subgraph of G_f with only the edges with positive residual capacity.
  - augmenting path = any s-t path in G_f^+

- **Level $\ell(v)$ of node v:**
  length (# of edges) of shortest path from s to v in G_f^+.

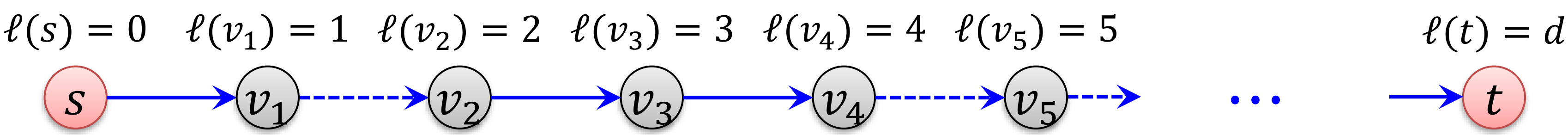


Shortest Augmenting Path Algorithm

Lemma 1: For every node v, the level $\ell(v)$ is non-decreasing.

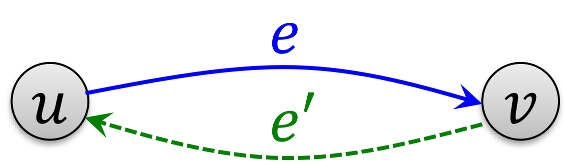
Proof:

- Consider augmentation along one augmenting path



  - Before augmentation, edges are between consecutive levels

- The set of edges of G_f^+ only changes if the residual capacity of some edge changes:



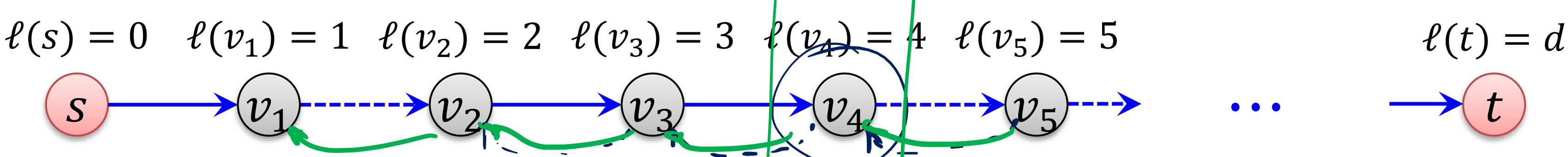
  - If e is on augmenting path P and $c_e = \text{bottleneck}(P, f)$, after augmentation, $c_e = 0$ and e is removed from G_f^+
  - The residual cap. of the edge e' in the opposite direction could increase from 0 to > 0 and be added to G_f^+

Shortest Augmenting Path Algorithm

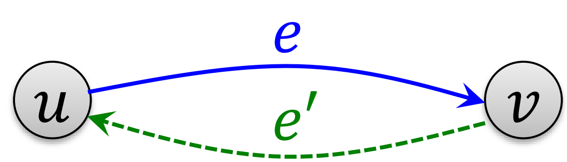
Lemma 1: For every node v, the level $\ell(v)$ is non-decreasing.

Proof:

- Consider augmentation along one augmenting path



  - Before augmentation, edges are between consecutive levels
  - A shortest augmenting path consists of exactly one node of each level.
  - The only new edges are from level $i + 1$ to level i for some $i \geq 0$.
    (for the levels before augmenting along the path)



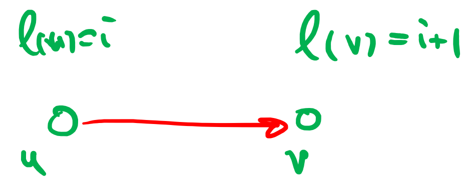
  - Such edges cannot create shortcuts to create s-w paths of length $< \ell(w)$
  - Levels of all nodes are non-decreasing.

Shortest Augmenting Path Algorithm

Lemma 2: There are at most $O(m \cdot n)$ augmentation steps.

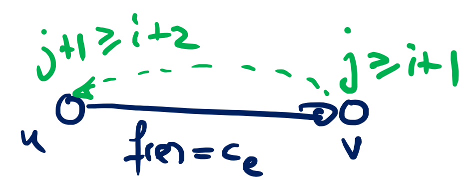
Proof:

- In each augmentation step, at least one edge (u, v) is deleted from G_f^+
  - Some edge $e = (u, v)$ on the augmenting path P has $c_e = \text{bottleneck}(P, f)$
  - The residual capacity of e is set to 0 and e is removed from G_f^+



- When (u, v) is deleted from G_f^+, for some $i \geq 0$:

$$\ell(u) = i, \qquad \ell(v) = i + 1$$



- If (u, v) is later added back to G_f^+, for some $j \geq 0$:

$$\ell(u) = j + 1, \qquad \ell(v) = j$$

- Because level $\ell(v)$ is non-decreasing: $j \geq i + 1$
  $\Longrightarrow$ When (u, v) is added back, $\ell(u) \geq i + 2$

- Because the maximum possible level is $n - 1$, each edge is deleted from G_f^+ at most $O(n)$ times.

Shortest Augmenting Path Algorithm

Theorem: The Edmonds-Karp algorithm computes a maximum flow in time $O(m^2 n)$ even with arbitrary non-negative capacity values.

- *Edmonds-Karp algorithm = Ford-Fulkerson algorithm, where we choose a shortest augmenting path in each step.*

Proof:

- From lemma before: $O(\underline{m \cdot n})$ augmentation steps
- A shortest augmenting path can be found in time $O(\underline{m + n})$ by using a BFS traversal on the positive residual graph G_f^+.

Other Algorithms

- There are many other algorithms to solve the maximum flow problem, for example:

- **Preflow-push algorithm:** [Goldberg,Tarjan 1986]
  - Maintains a preflow (∀ nodes: inflow ≥ outflow)
  - Alg. guarantees: As soon as we have a flow, it is optimal
  - Detailed discussion in 2012/13 lecture
  - Running time of basic algorithm: $O(m \cdot n^2)$
  - Doing steps in the "right" order: $O(n^3)$

- **Current best known complexity:** $\boldsymbol{O(\underline{m \cdot n})}$
  - For graphs with $m \geq n^{1+\epsilon}$ [King,Rao,Tarjan 1992/1994]
    (for every constant $\epsilon > 0$)
  - For sparse graphs with $m \leq n^{16/15-\delta}$ [Orlin, 2013]

- **Best known since 2022:** $\boldsymbol{O\left(\underline{m \cdot n^{o(1)}}\right)}$ $O(m^{1+o(1)})$
  - Uses a continuous optimization approach
  - Published by Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva