Chapter 6
Graph Algorithms

Algorithm Theory
WS 2017/18

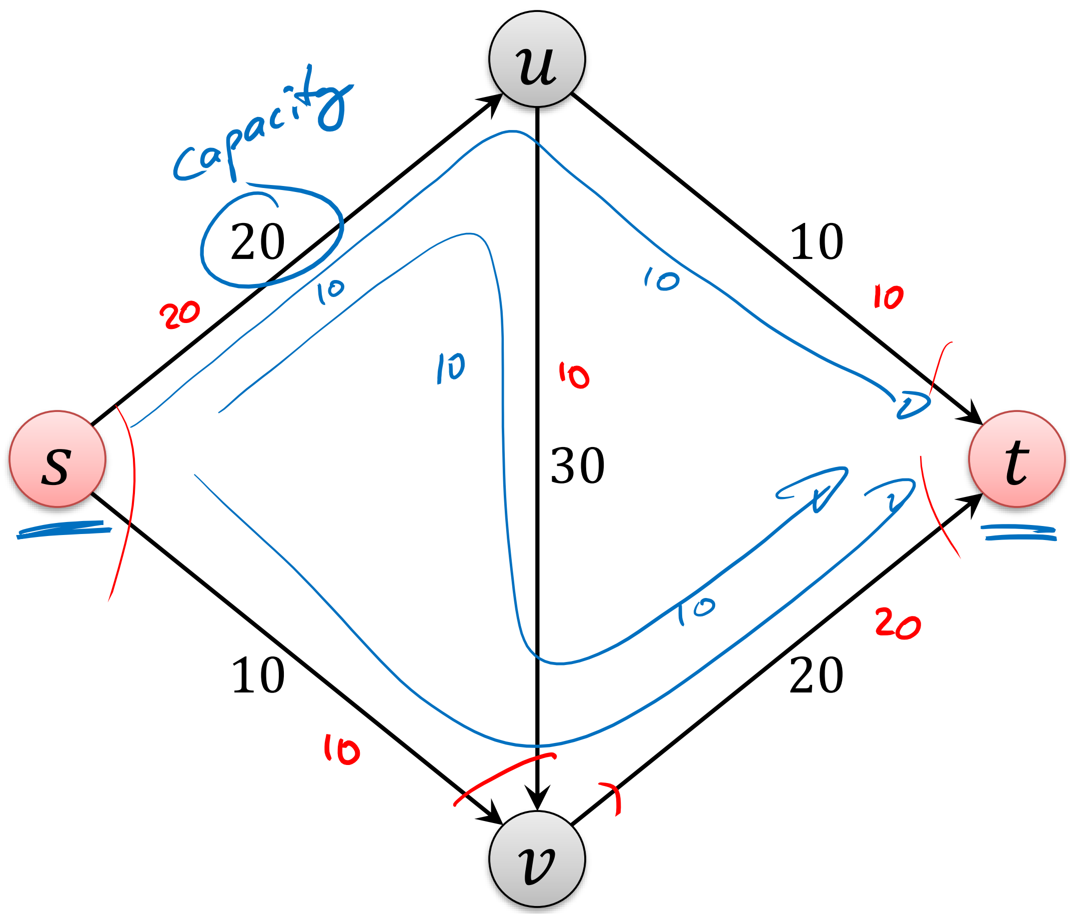
next week

lecture : Mon, Dec 4

exercises: Thu, Dec 7

Fabian Kuhn

Example: Flow Network



Network Flow: Definition

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

- $f(e)$ is the amount of flow carried by edge e

Capacity Constraints:

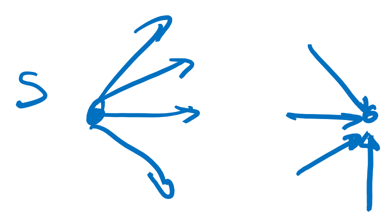
$\sum_{v \in V} f_{out}(v) = \sum_{v} f_{in}(v)$

- 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:

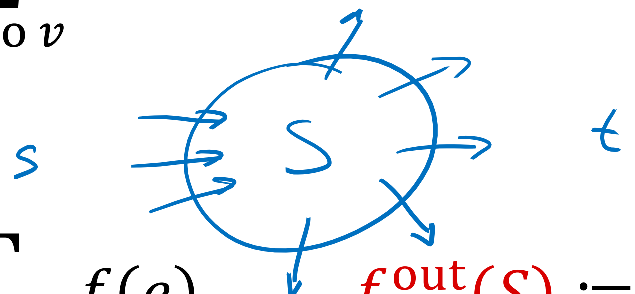
$$|f| := \sum_{e \text{ out of } s} f\big((s,u)\big) = \sum_{e \text{ into } t} f\big((v,t)\big)$$

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 $S \subseteq V$:



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

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

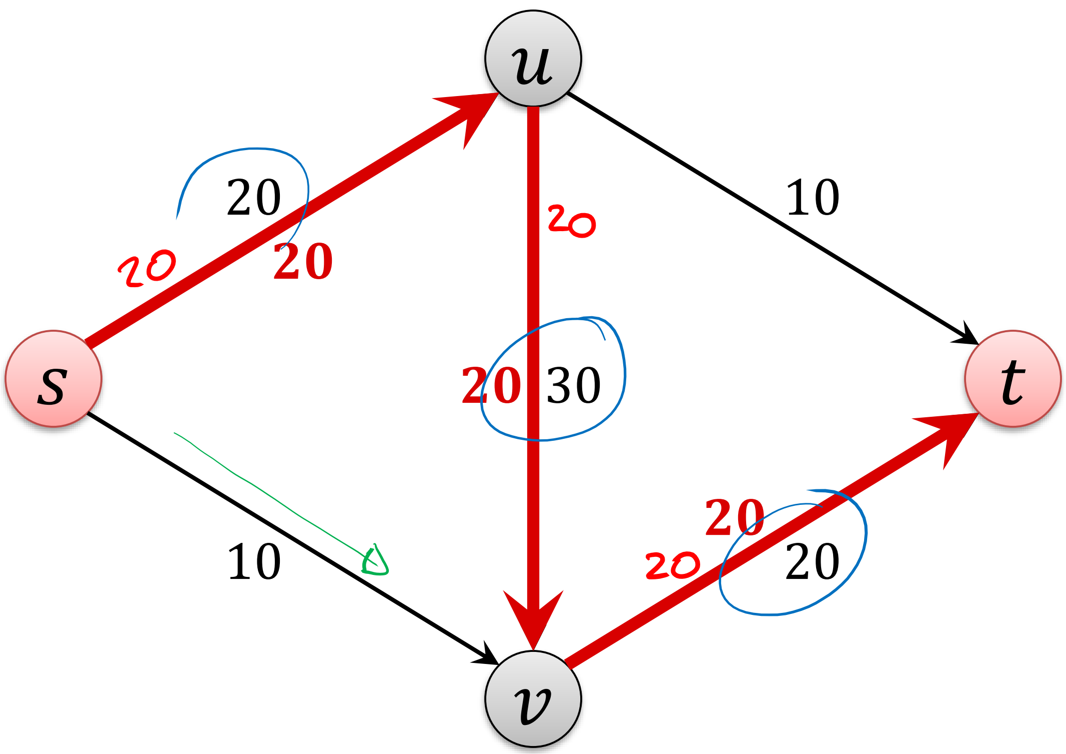
- Classical graph optimization problem
- Many applications (also beyond the obvious ones)
- Requires new algorithmic techniques

Maximum Flow: Greedy?

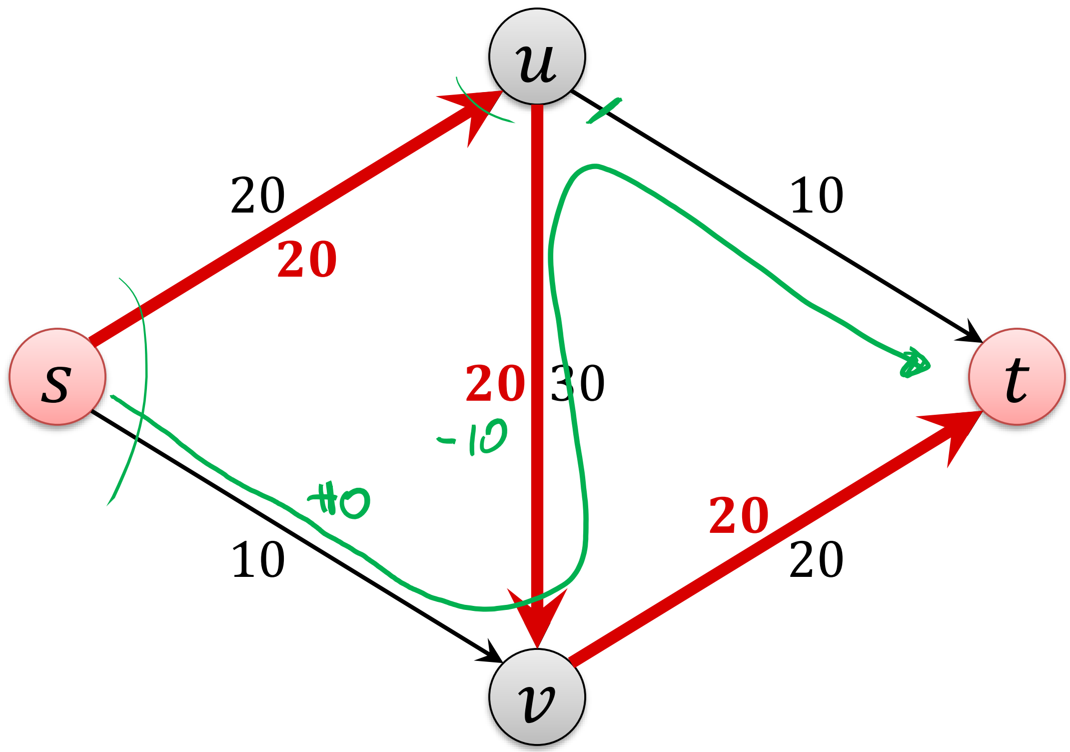
Does greedy work?

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)

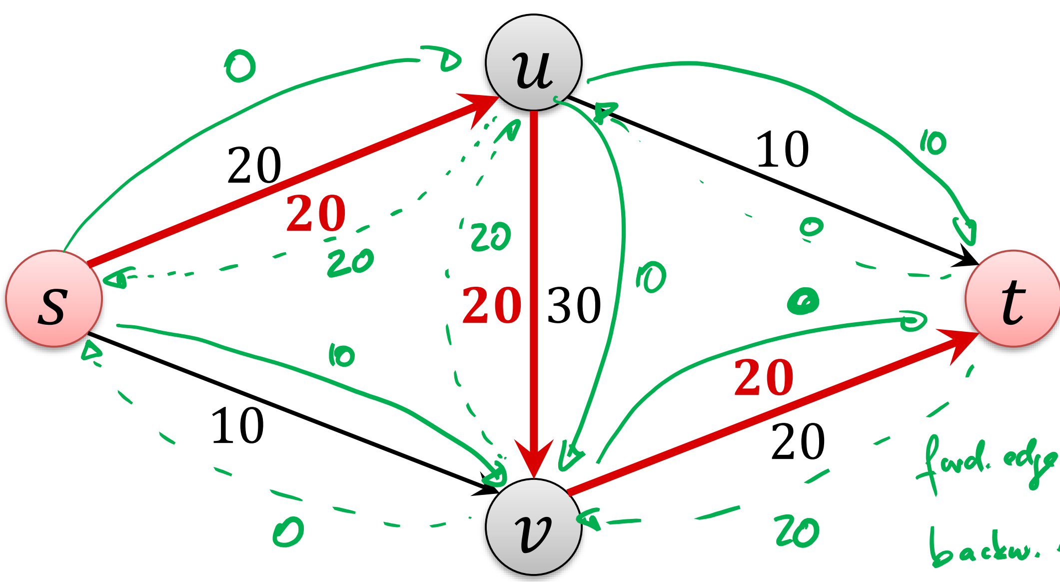
Residual Graph

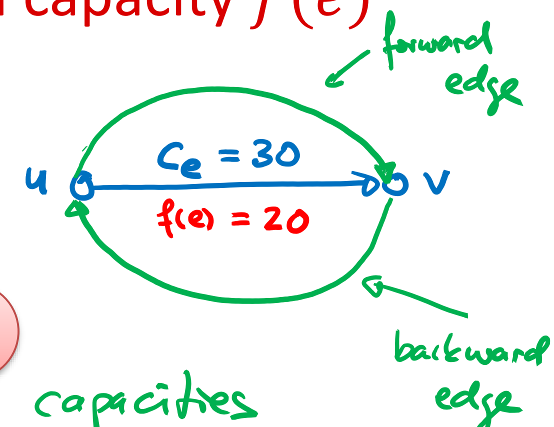
Given a flow network $G = (V, E)$ with capacities c_e (for $e \in E$)

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

residual graph

- 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 $f(e)$



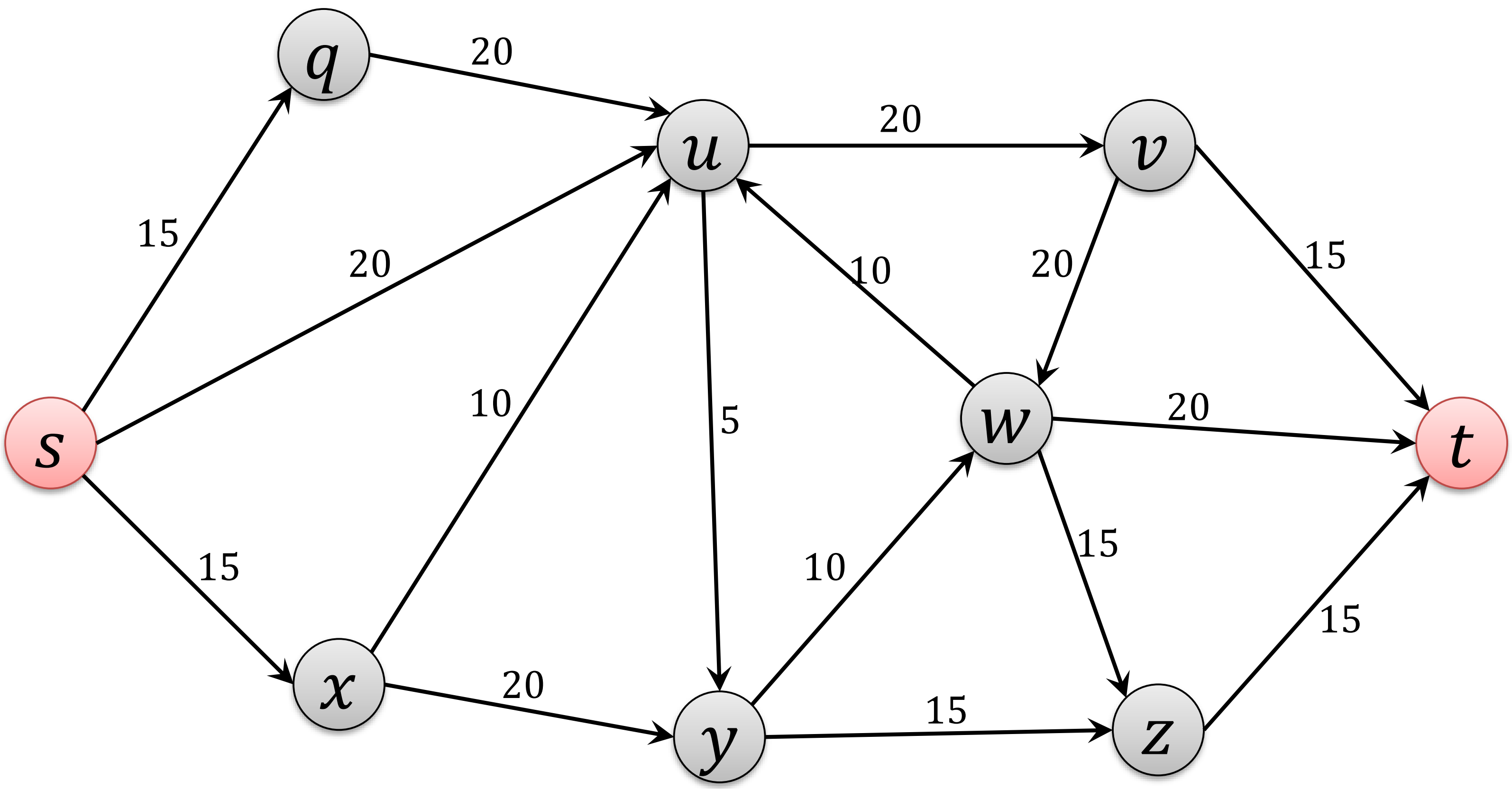


capacities

fwd. edge: $cap = c_e - f(e) = 10$

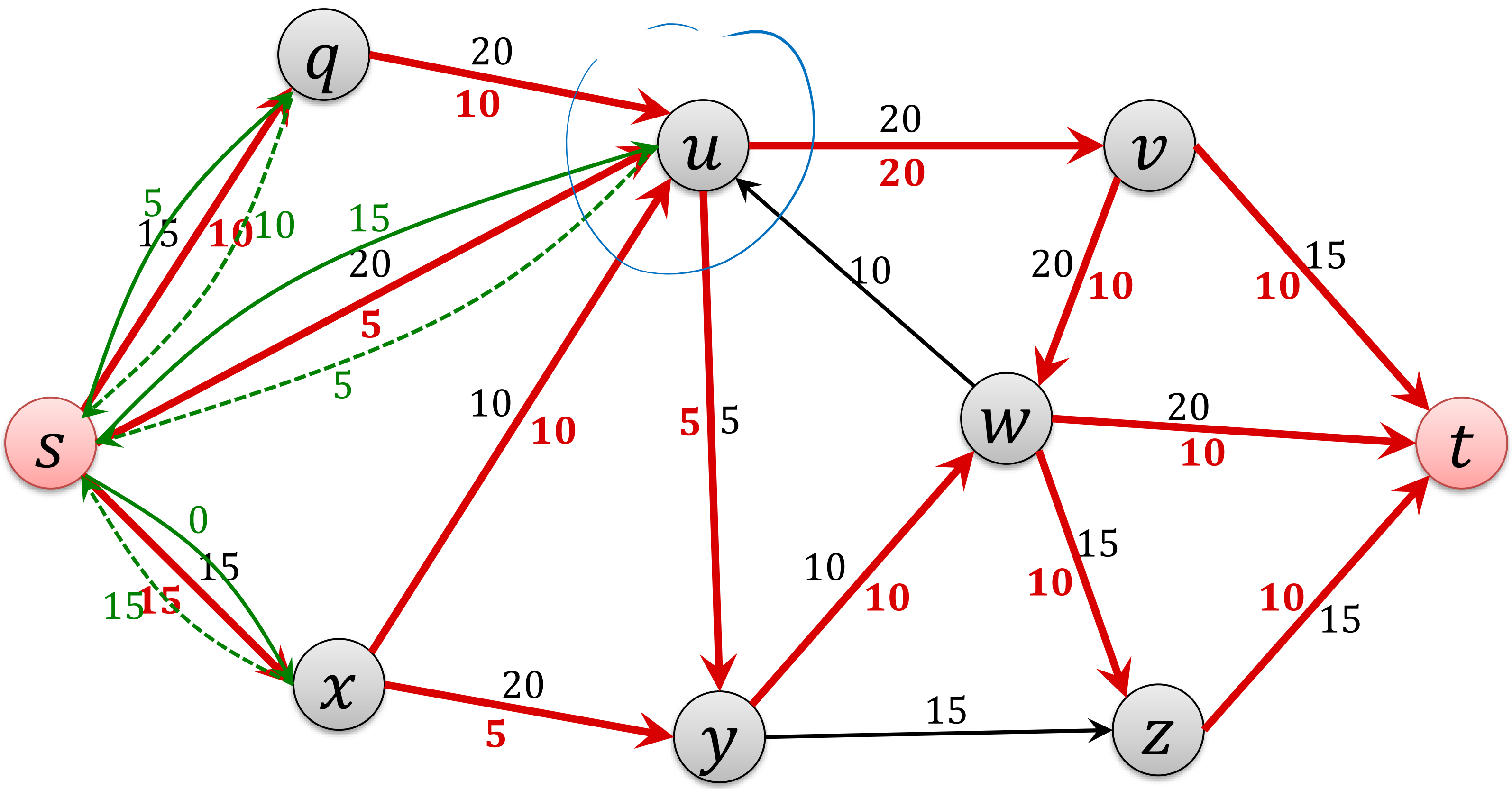
backw. edge: $cap = f(e) = 20$

Residual Graph: Example



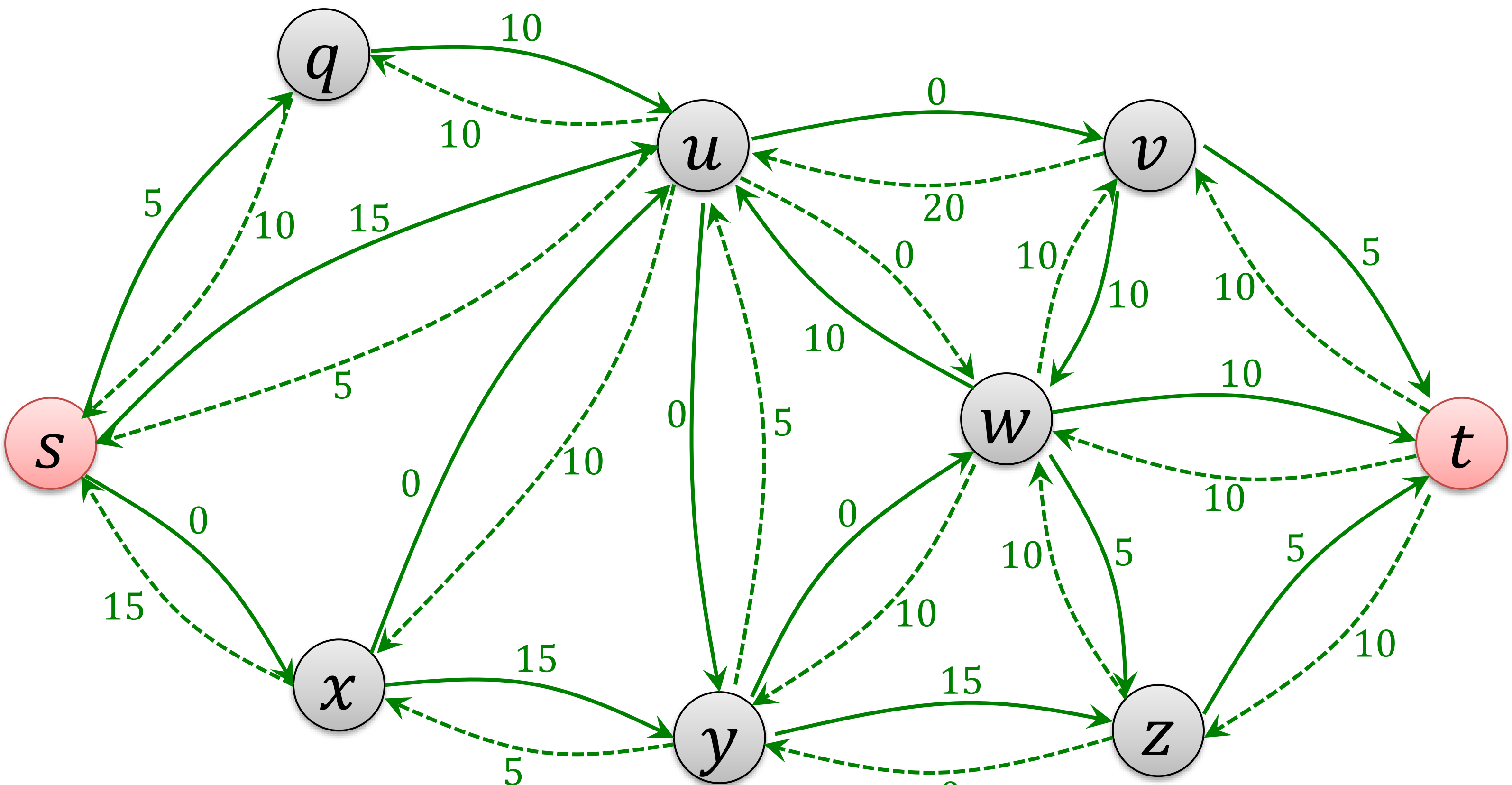
Residual Graph: Example

Flow f



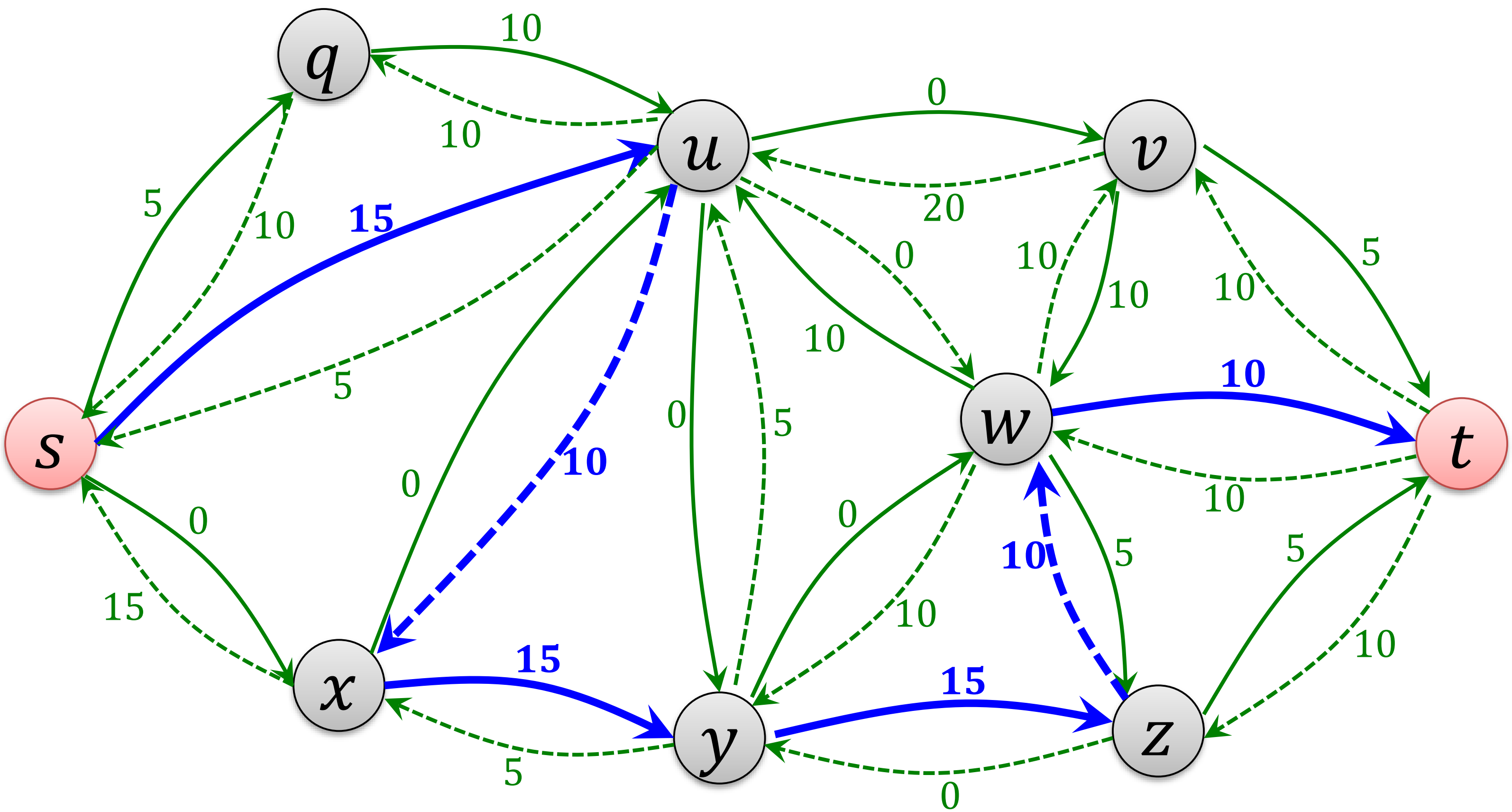
Residual Graph: Example

Residual Graph G_f



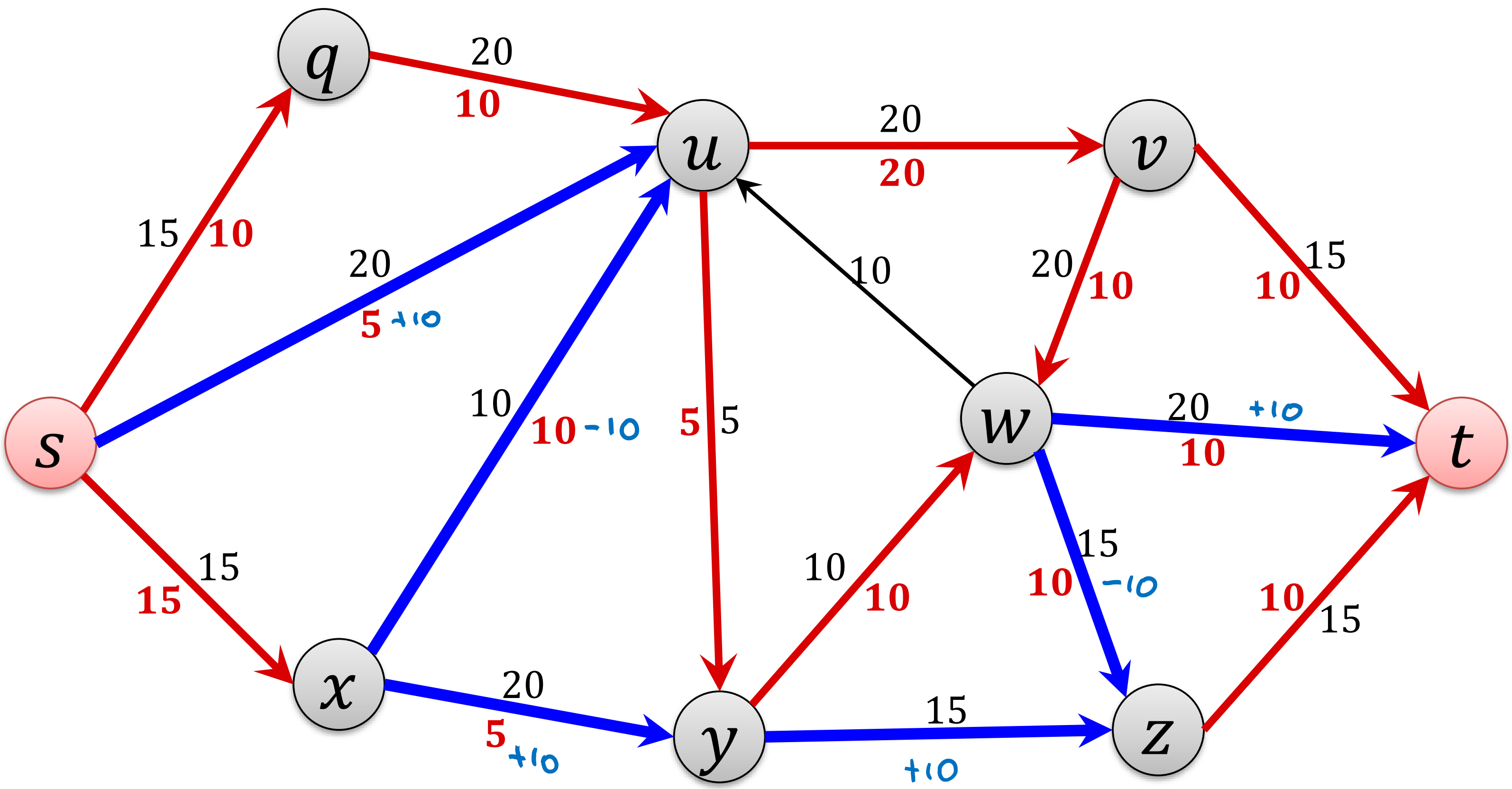
Augmenting Path

Residual Graph G_f



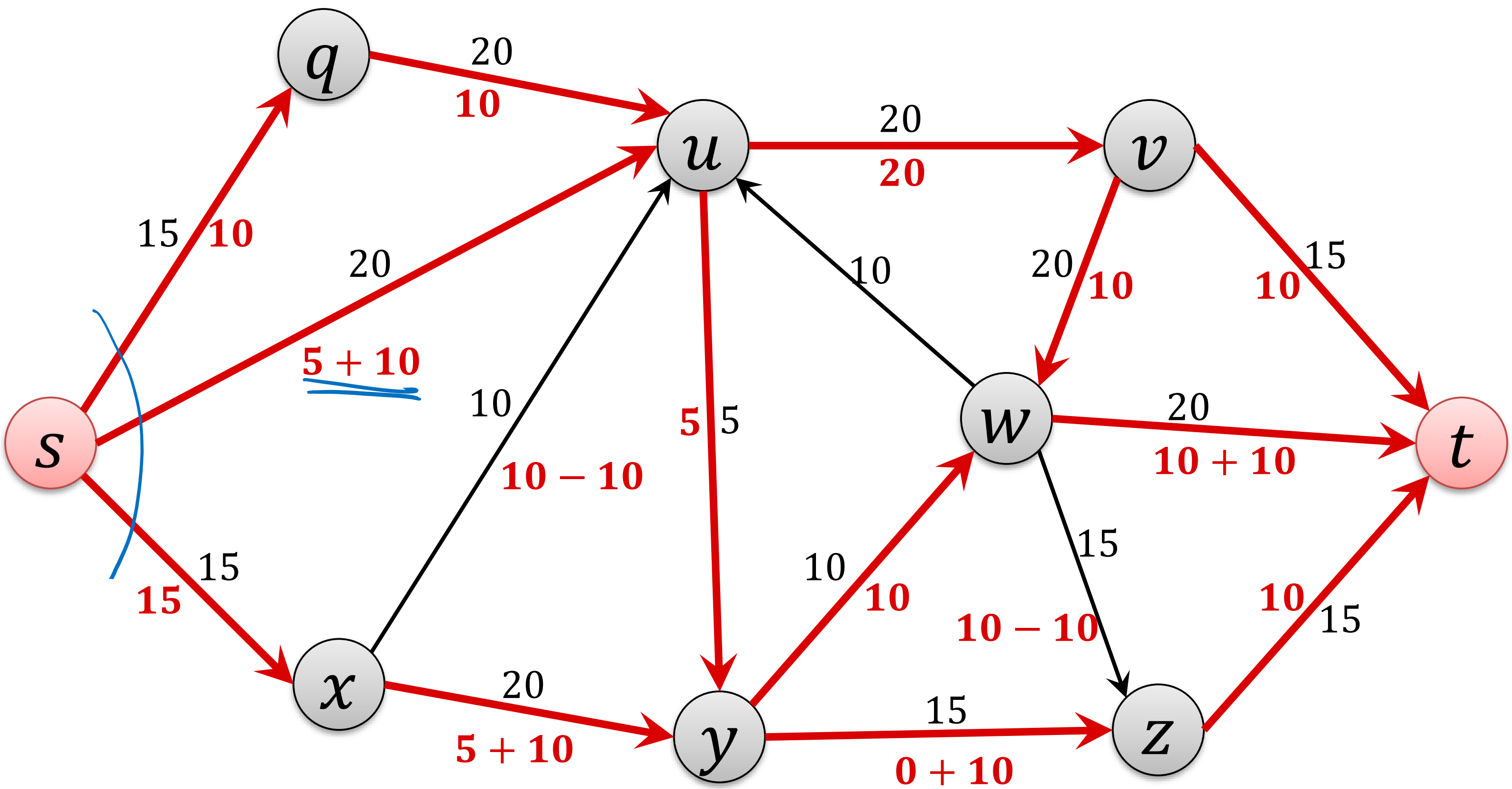
Augmenting Path

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:

$$f'\big((u,v)\big) := f\big((u,v)\big) + \text{bottleneck}(P, f)$$

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

$$f'\big((v,u)\big) := f\big((v,u)\big) - \text{bottleneck}(P, f)$$

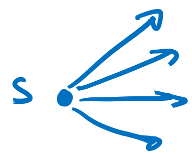
Augmented Flow

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

$$|f'| = |f| + \underbrace{\mathbf{bottleneck}(P, f)}_{b}.$$

Proof:

$|f'| = |f| + b$ ✓

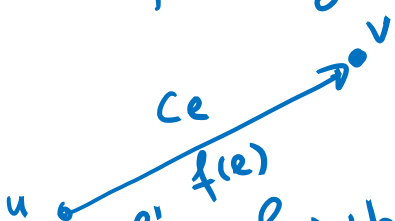


P leaves s on a fwd. edge

$\underline{f' \text{ is legal}}$ $\forall e \in E: \quad 0 \le f'(e) \le c_e \quad$ (I) ✓

$\forall v \in V \setminus \{s,t\}: \quad f^{in}(v) = f^{out}(v) \quad$ (II)

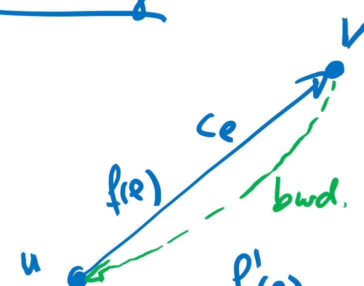
$\underline{(I):}$ fwd. edge



$f'(e) = f(e) + b \le c_e$

cap. of fwd. edge: $c_e - f(e) \ge b$

backw. edge



bwd. edge cap $= f(e)$

$f'(e) = f(e) - b$

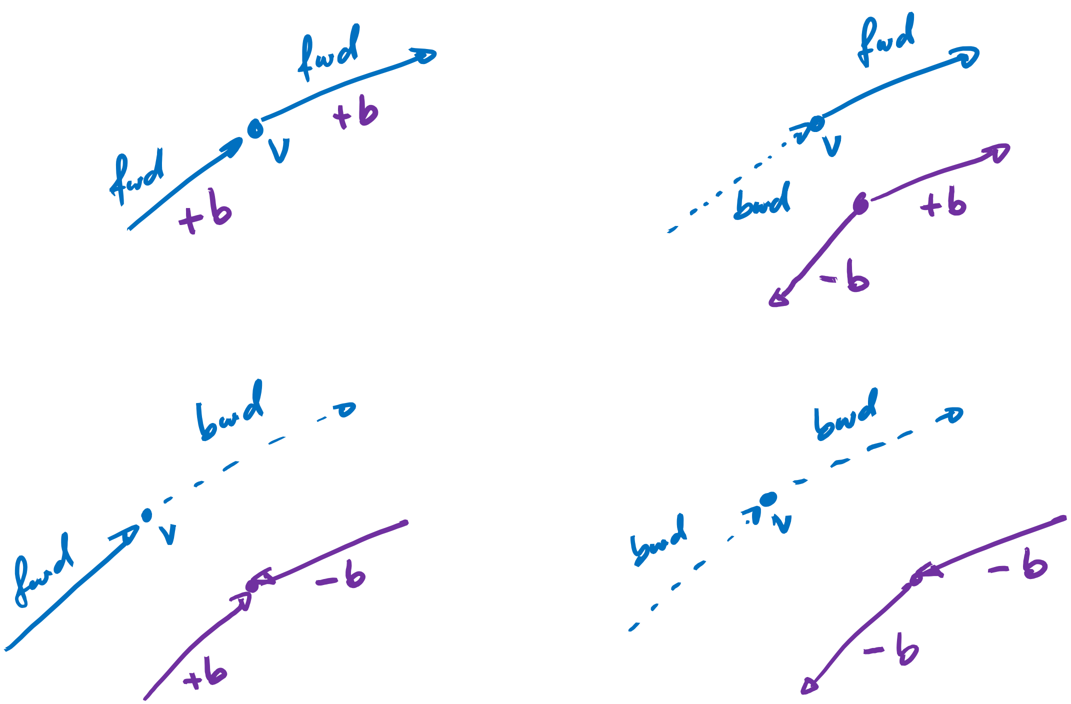
Augmented Flow

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

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

Proof:

flow conservation (consider $v \neq s,t$ on P)



□

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**;

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

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:

At all times, for all $e \in E$, $f(e)$ is an integer

initially $f(e) = 0$

in one iter. augm. P: residual cap. are integers

bottleneck$(P,f) > 0$ (also bottleneck(P,f) is integer)

$\Longrightarrow$ bottleneck$(P,f) \geq 1$

$\longrightarrow$ new flow values are integers

$\longrightarrow |f'| \geq |f| + 1$

$\Longrightarrow \leq C$ iterations

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:

Claim: one iteration can be computed in $O(m)$ time

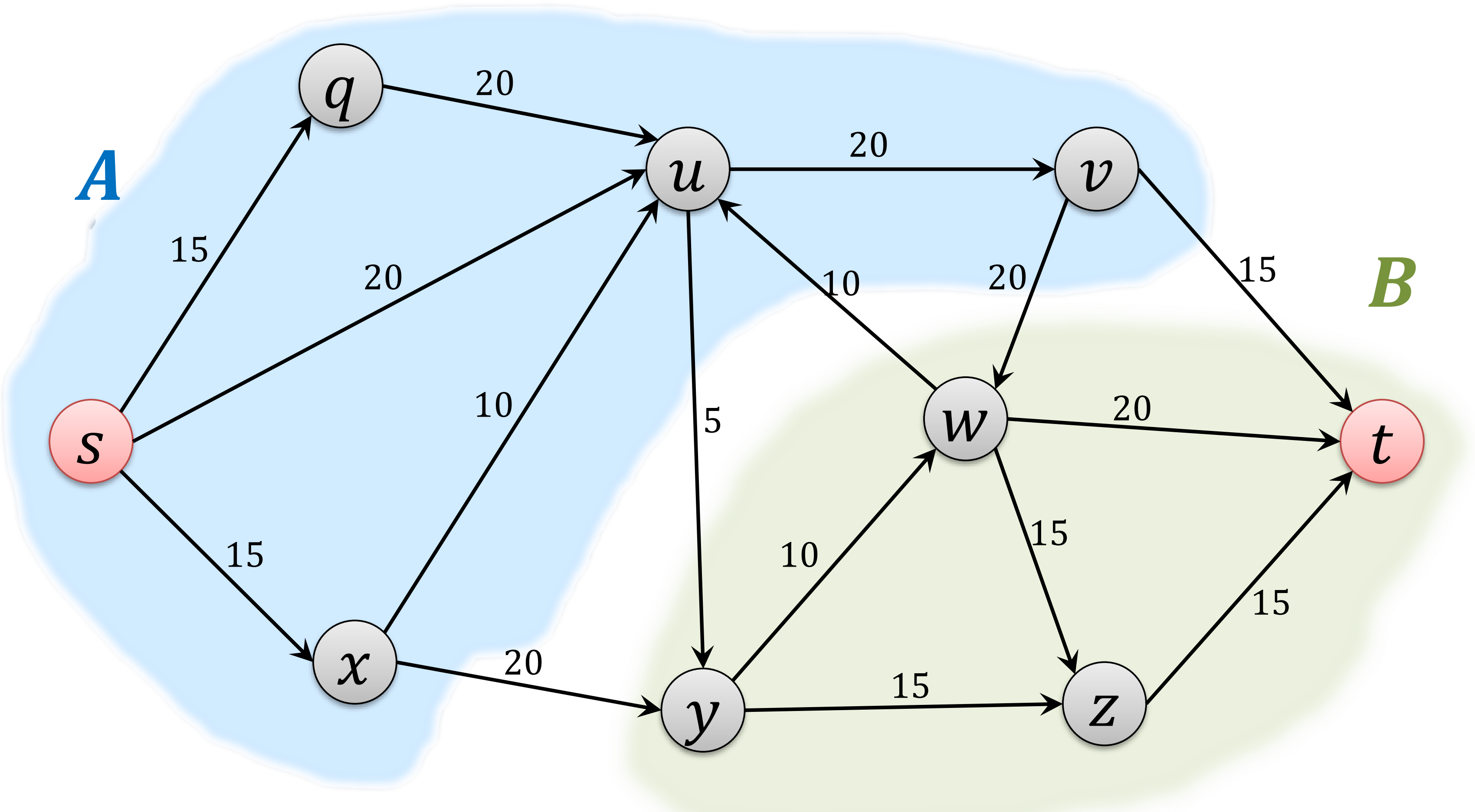
1. compute / update residual graph
   - first iter.: $O(m)$
   - later iter.: $O(n)$

2. find augm. path / conclude there is no augm. path
   - ↳ s-t path in G_f with res. cap. > 0
   - ↳ graph traversal (DFS/BFS) : $O(m)$ time

3. update flow values : $O(n)$

s-t Cuts

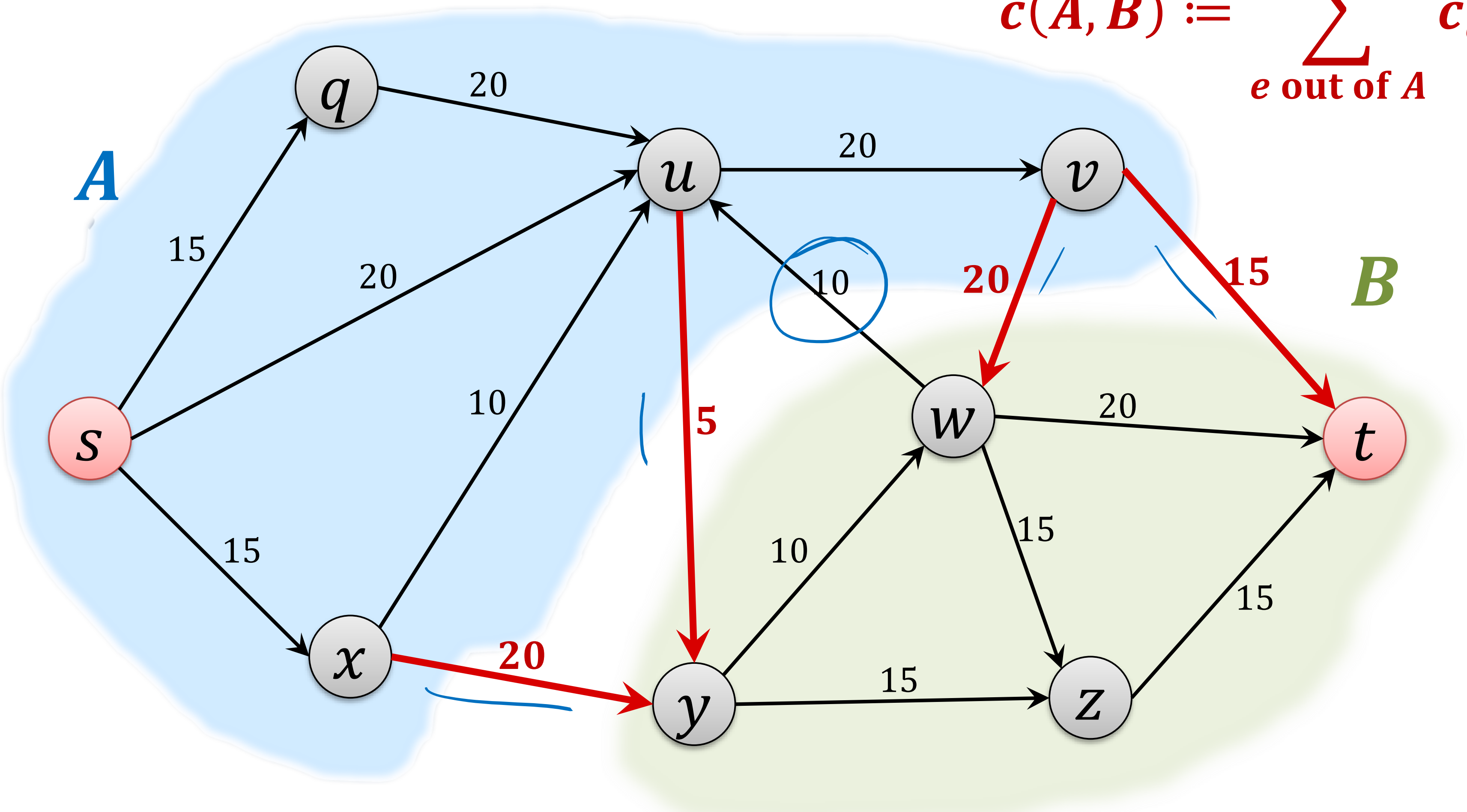
Definition:
An s-t cut is a partition (A, B) of the vertex set such that $s \in A$ and $t \in B$



Cut Capacity

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

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



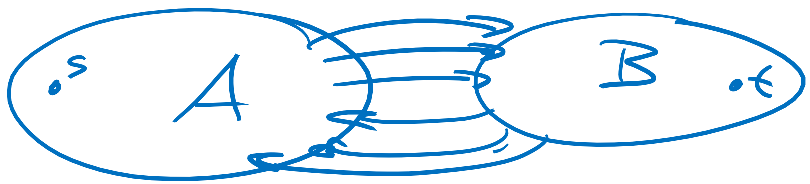
Cuts and Flow Value

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:

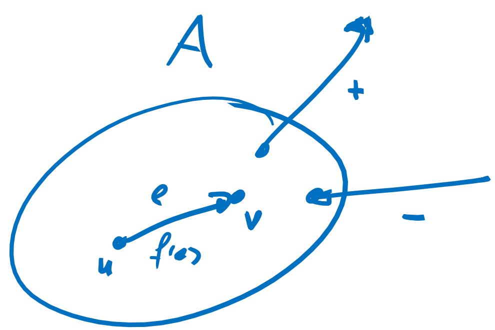
$|f| = f^{out}(s) \quad (= f^{in}(t))$



$$|f| = f^{out}(s) - f^{in}(s)$$

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

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



Cuts and Flow Value

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

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

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

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

Proof:

either do symmetric argument

<u>or:</u> $f^{\text{out}}(A) = f^{\text{in}}(B)$

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

Upper Bound on Flow Value

Lemma:

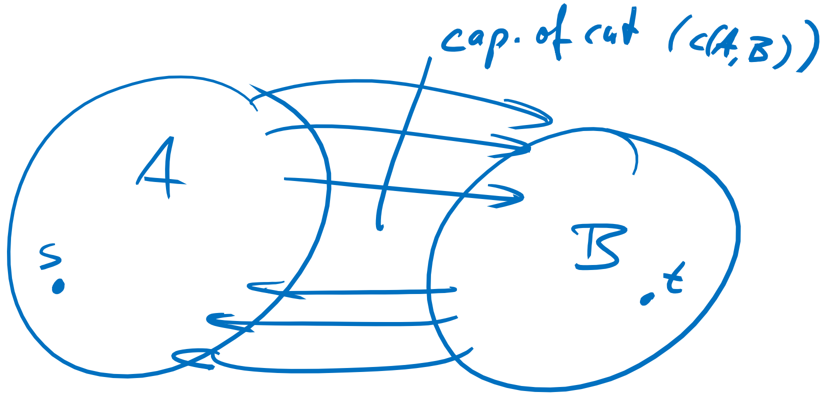
Let f be any s-t flow and (A, B) any s-t cut. Then $\boldsymbol{|f| \leq c(A, B)}$.

Proof:

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

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

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



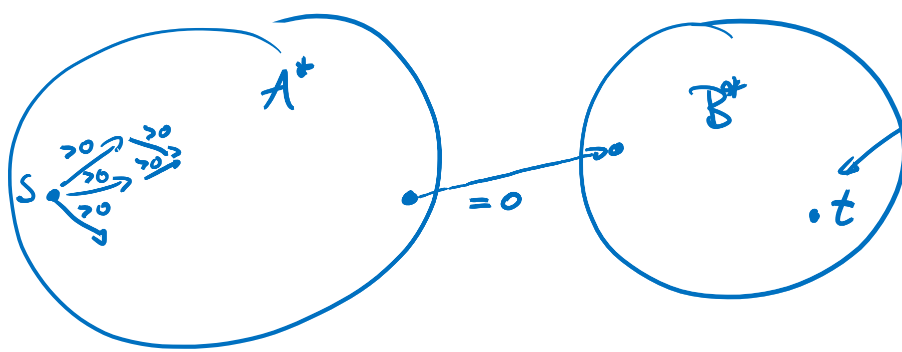
Ford-Fulkerson Gives Optimal Solution

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

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

Proof:

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



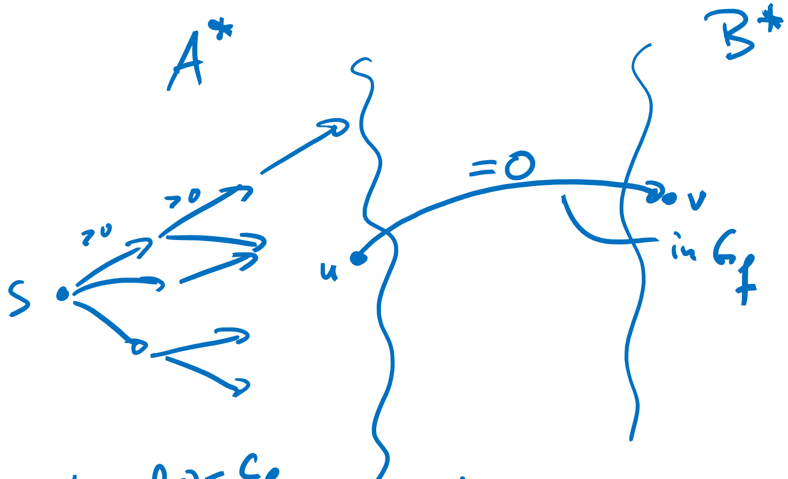
t can not be reached on only edges with pos. res. cap. in G_f

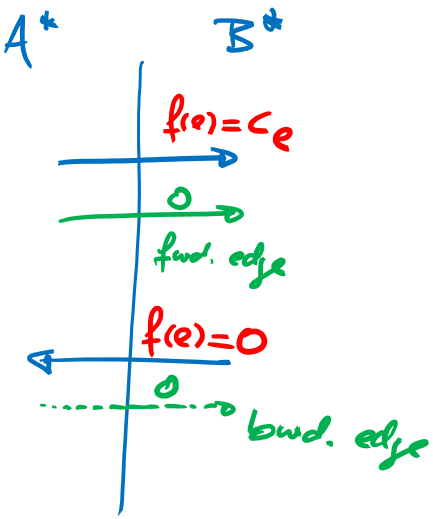
- For $B^* = V \setminus A^*$, (A^*, B^*) is an s-t cut
  - By definition $s \in A^*$ and $t \notin A^*$

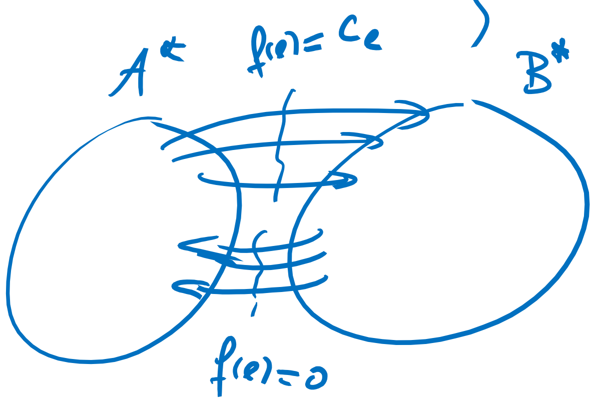
Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

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

Proof:







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

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

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

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

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

Proof:

Ford-Fulkerson Gives Optimal Solution

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

Proof:

FF gives a flow f^* and a cut (A^*, B^*)

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

we have seen that $|f| \leq c(A^*, B^*)$

for every legal flow

Min-Cut Algorithm

s-t cut of minimum cap.

Ford-Fulkerson also gives a min-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 augm. path

can find cut (A^*, B^*) s.t. $|f| = c(A^*, B^*)$

↳ as before using DFS / BFS

(A^*, B^*) is an s-t cut of min capacity

because: for every other s-t cut (A, B)

we know that $|f| \leq c(A, B)$

$\Downarrow$

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

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:

FF gives f^* cut (A^*, B^*)

s.t. f^* maximum flow

(A^*, B^*) min. s-t-cut

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

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:

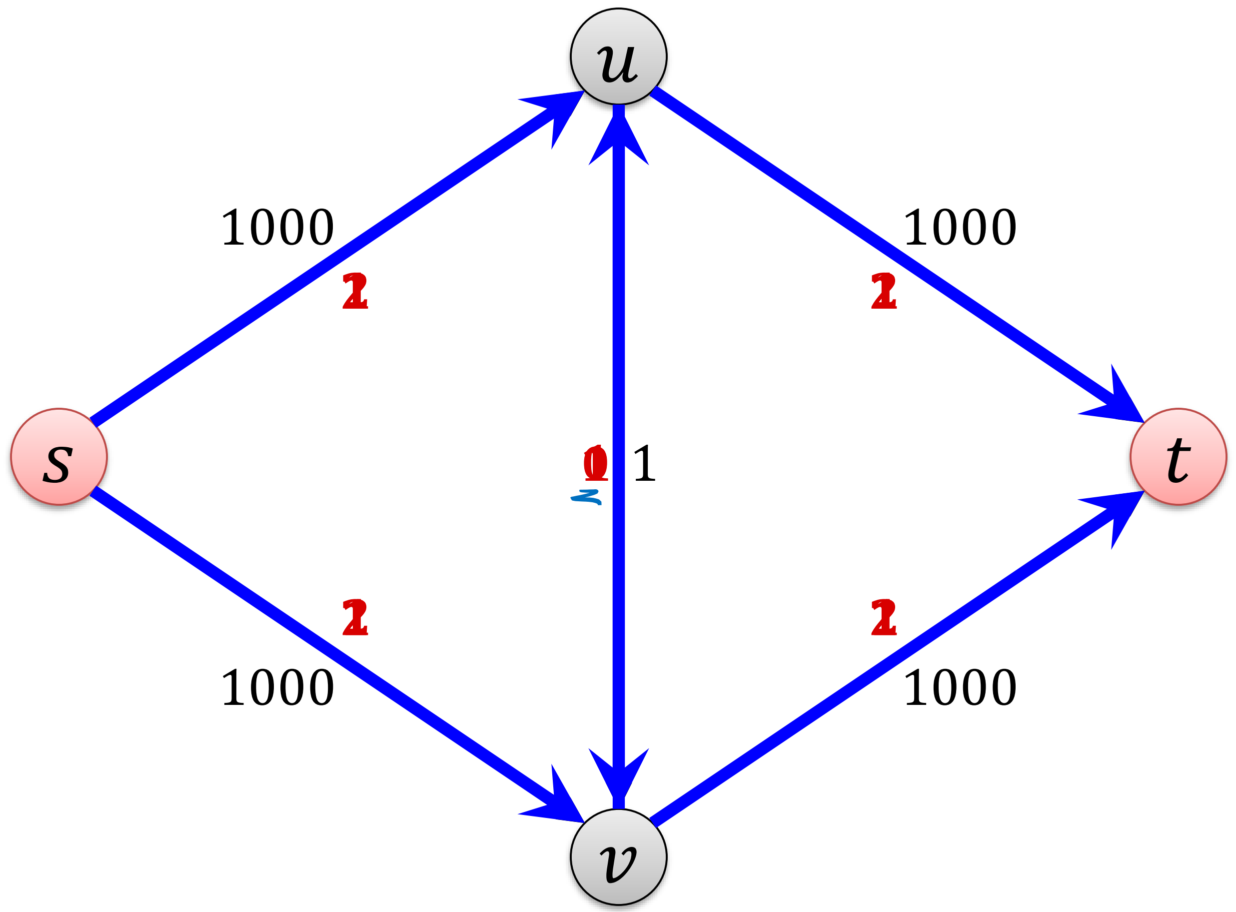
FF gives an int. sol.

Non-Integer Capacities

What if capacities are not integers?

- rational capacities: $\in \mathbb{Q}$
  - 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

- **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

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

$C := \underbrace{\max_e c_e}_{c_{max}}$

initially

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

# of scaling params: $\lfloor \log c_{max} \rfloor + 1$

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

running:

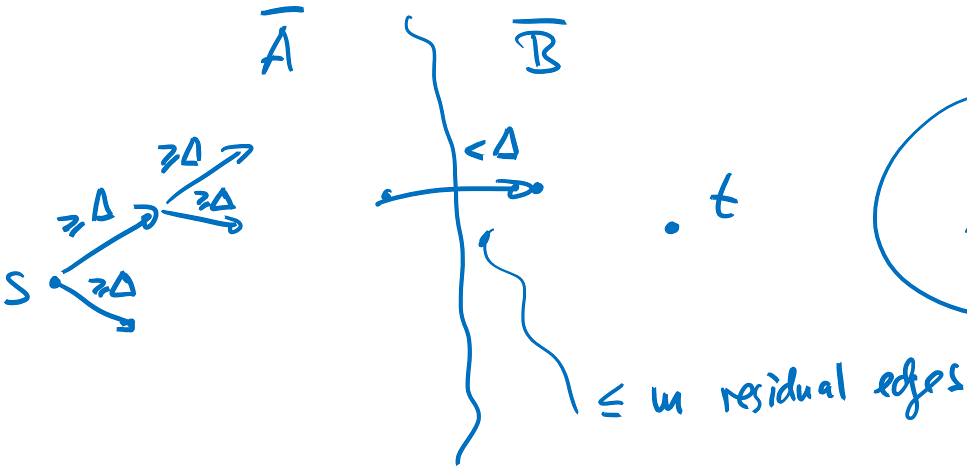
$\underbrace{\#\text{ phases}}_{O(\log C)} \cdot \underbrace{\#\text{ iter. per phase}}_{?} \cdot O(m)$

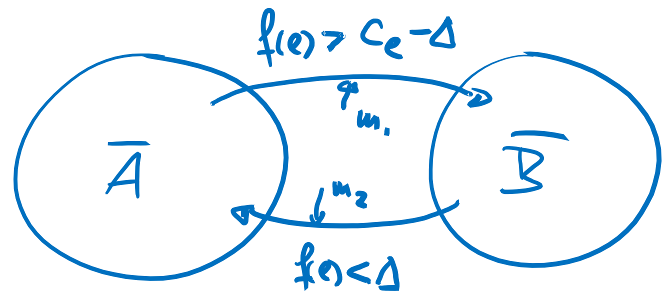
Length of a Scaling Phase

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

$|f^*| < |f| + m \cdot \Delta$

define cut $(\bar{A}, \bar{B})$





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

Length of a Scaling Phase

Lemma: The number of augmentation in each scaling phase is at most $2m$.

at the beginning of the Δ-scaling phase

↳ at the end of the 2Δ-scaling phase

$\Longrightarrow |f^*| < |f| + 2m\Delta$ (prev. Lemma)

each augm. path improves flow by Δ

$\Longrightarrow \leq 2m$ augm. in Δ-scaling phase

running time: $O(\log C) \cdot O(m) \cdot O(m) = O(m^2 \log C)$

Running Time: Scaling Max Flow Alg.

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

Strongly Polynomial Algorithm

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

$$\boxed{O(mC)}$$

- Time of algorithm with scaling parameter:

$$O(m^2 \log C)$$

- $O(\log C)$ is polynomial in the size of the input, but not in n
- Can we get an algorithm that runs in time polynomial in n?
- Always picking a shortest augmenting path leads to running time

$$\boxed{O(m^2 n)}$$

  – also works for arbitrary real-valued weights

non-negative

Other Algorithms

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

- **Preflow-push algorithm:**
  - Maintains a preflow ($\forall$ nodes: inflow $\geq$ 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: $O(m \cdot n)$**
  - For graphs with $m \geq n^{1+\epsilon}$ (for every constant $\epsilon > 0$) [King,Rao,Tarjan 1992/1994]
  - For sparse graphs with $m \leq n^{16/15-\delta}$ [Orlin, 2013]
  - approximate max flow in undirected graphs

$O(m \cdot n^{o(1)})$
necessary