



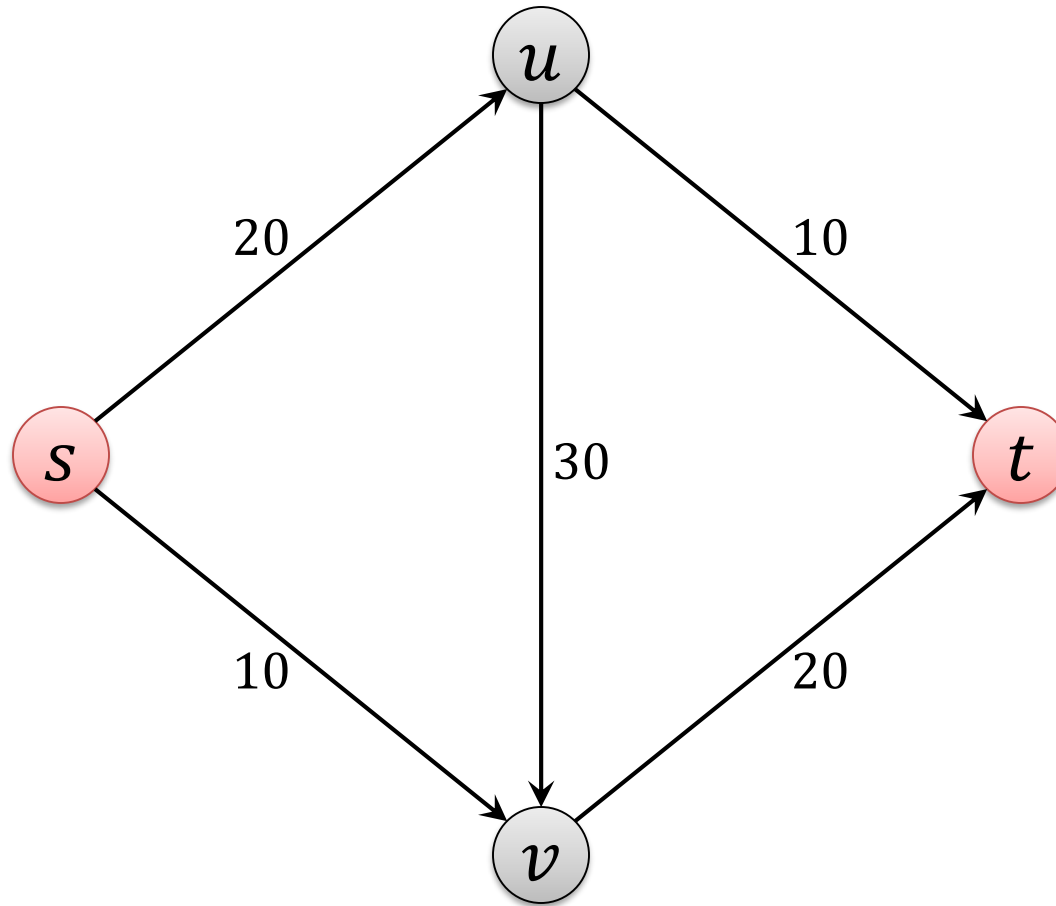
# **Chapter 6**

# **Graph Algorithms**

**Algorithm Theory**  
**WS 2015/16**

**Fabian Kuhn**

# Example: Flow Network



# Notation

We define:

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

For a set  $S \subseteq V$ :

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

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

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

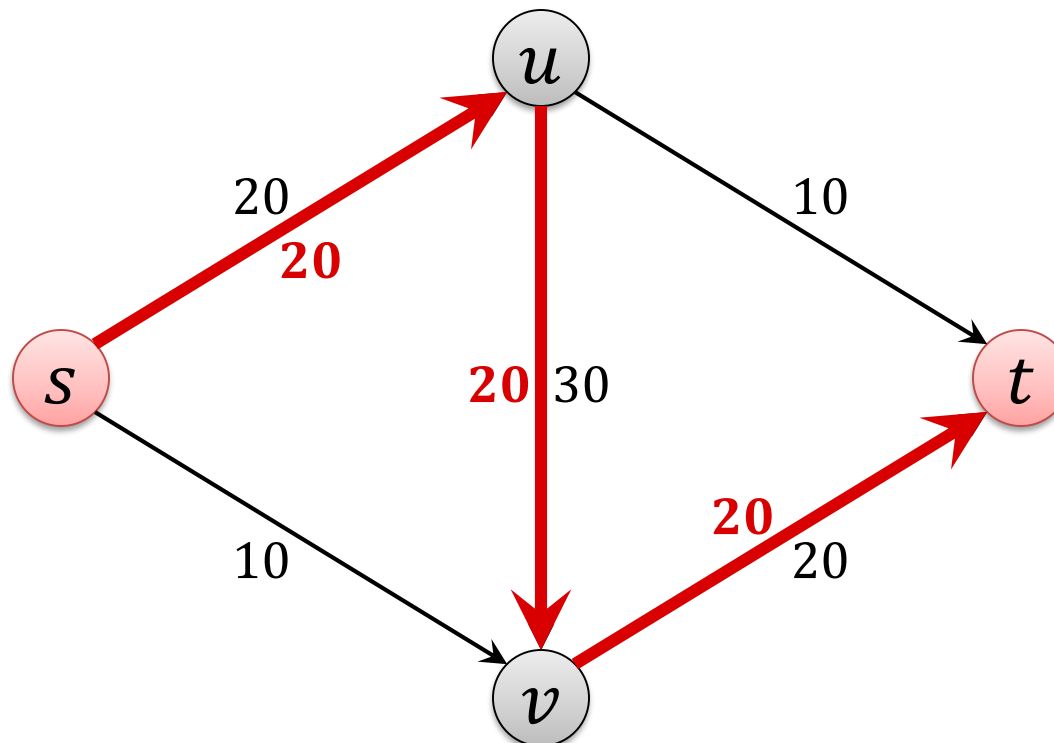
**For simplicity:** Assume that all **capacities** are **positive integers**

# Maximum Flow: Greedy?

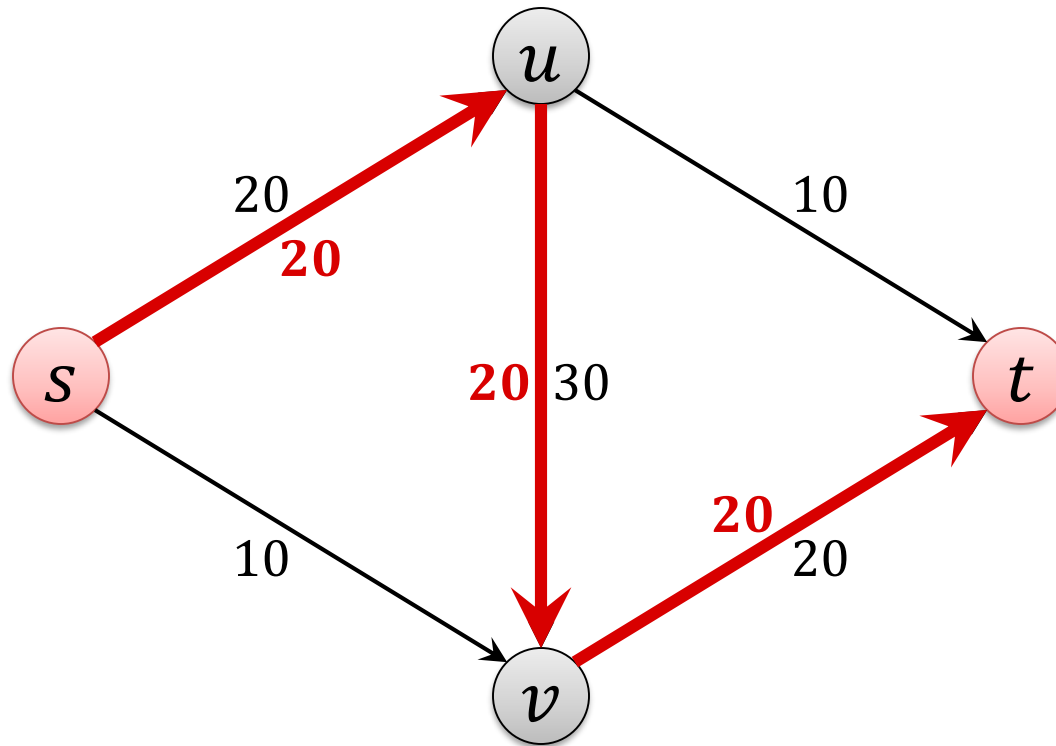
Does greedy work?

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



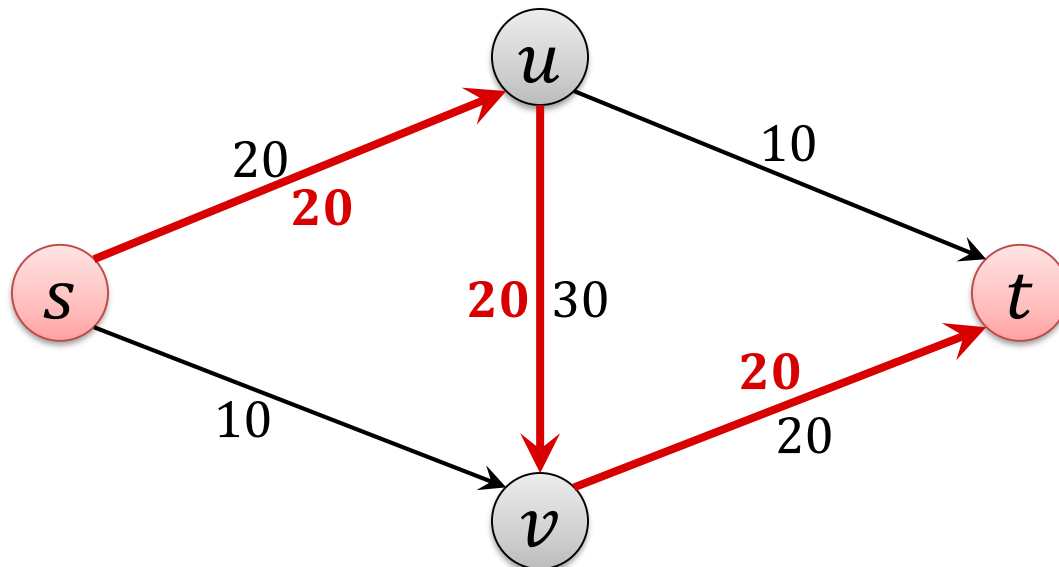
- Try to push 10 units of flow on edge  $(s, v)$
- Too much incoming flow at  $v$ : reduce flow on edge  $(u, v)$
- Add that flow on edge  $(u, t)$

# Residual Graph

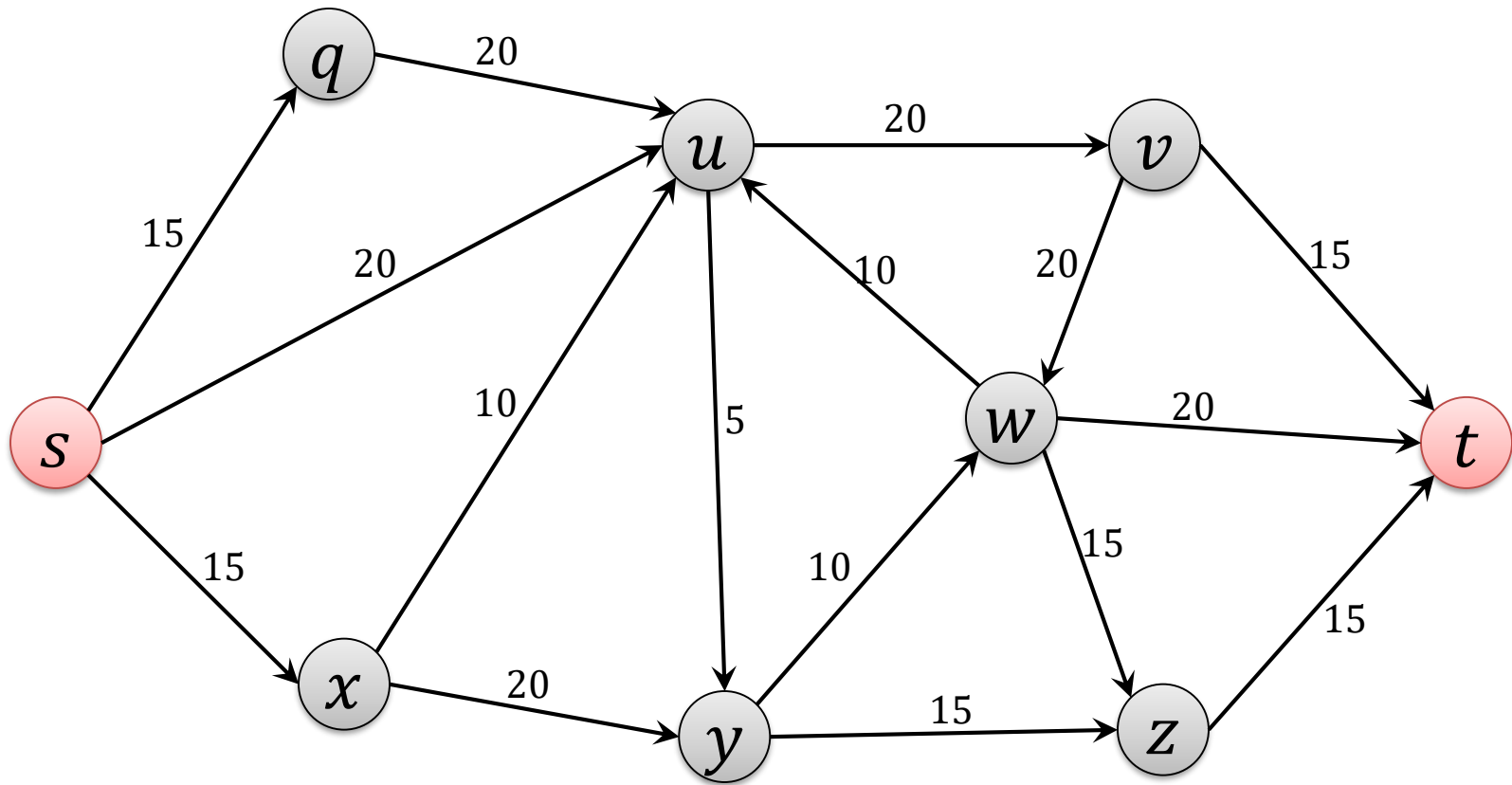
Given a flow network  $G = (V, E)$  with capacities  $c_e$  (for  $e \in E$ )

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**  $f(e)$

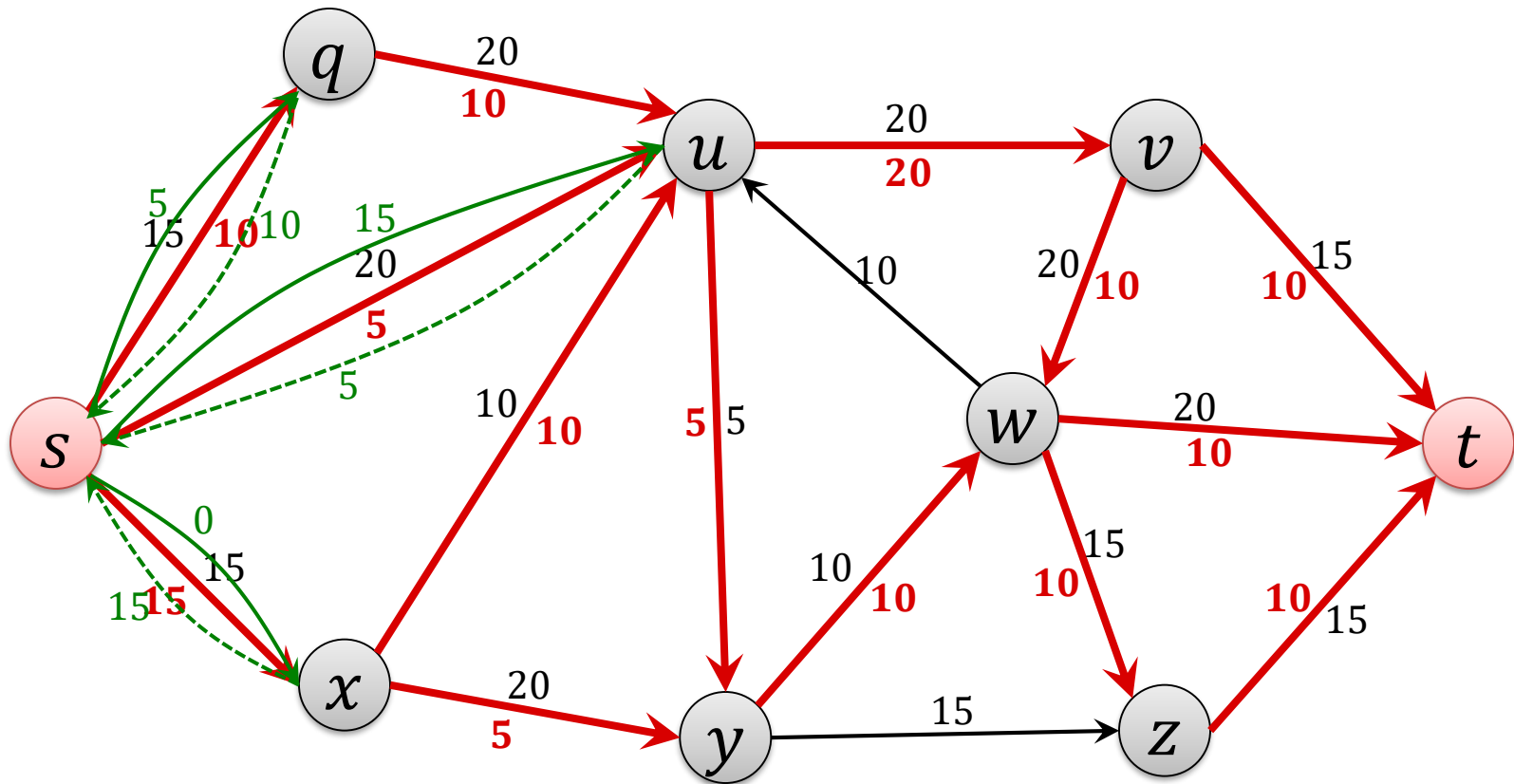


# Residual Graph: Example



# Residual Graph: Example

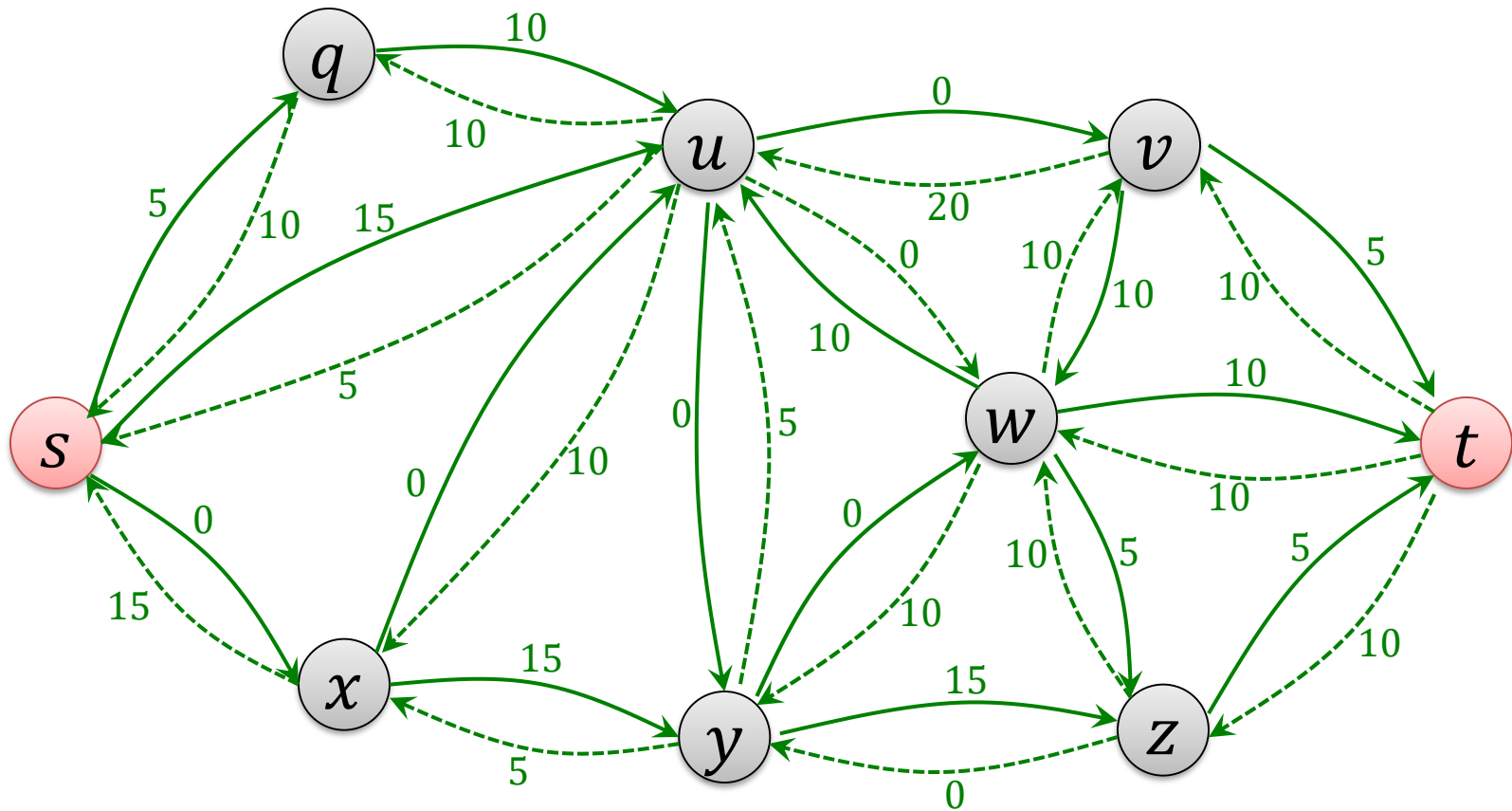
Flow  $f$





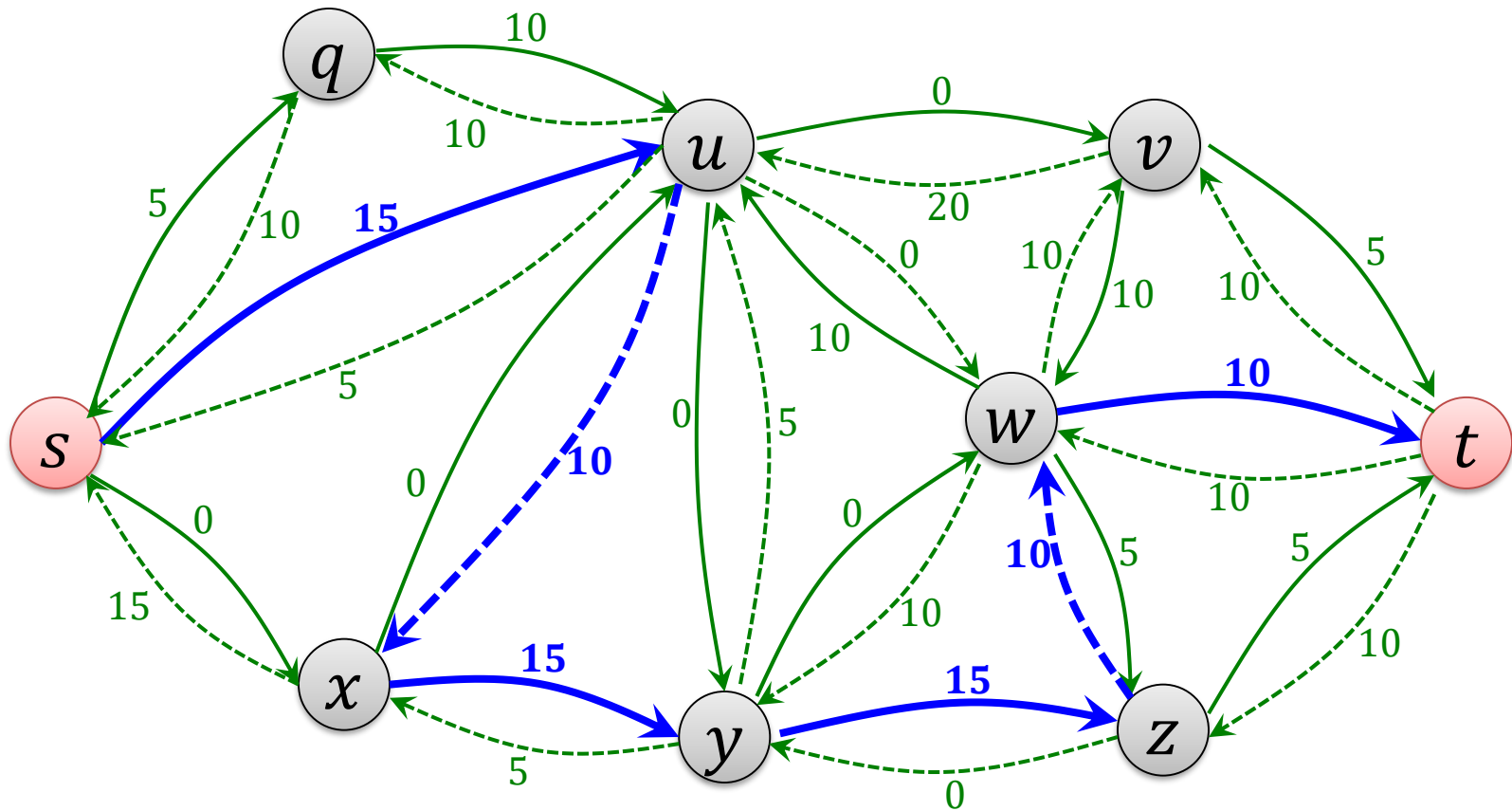
# Residual Graph: Example

## Residual Graph $G_f$



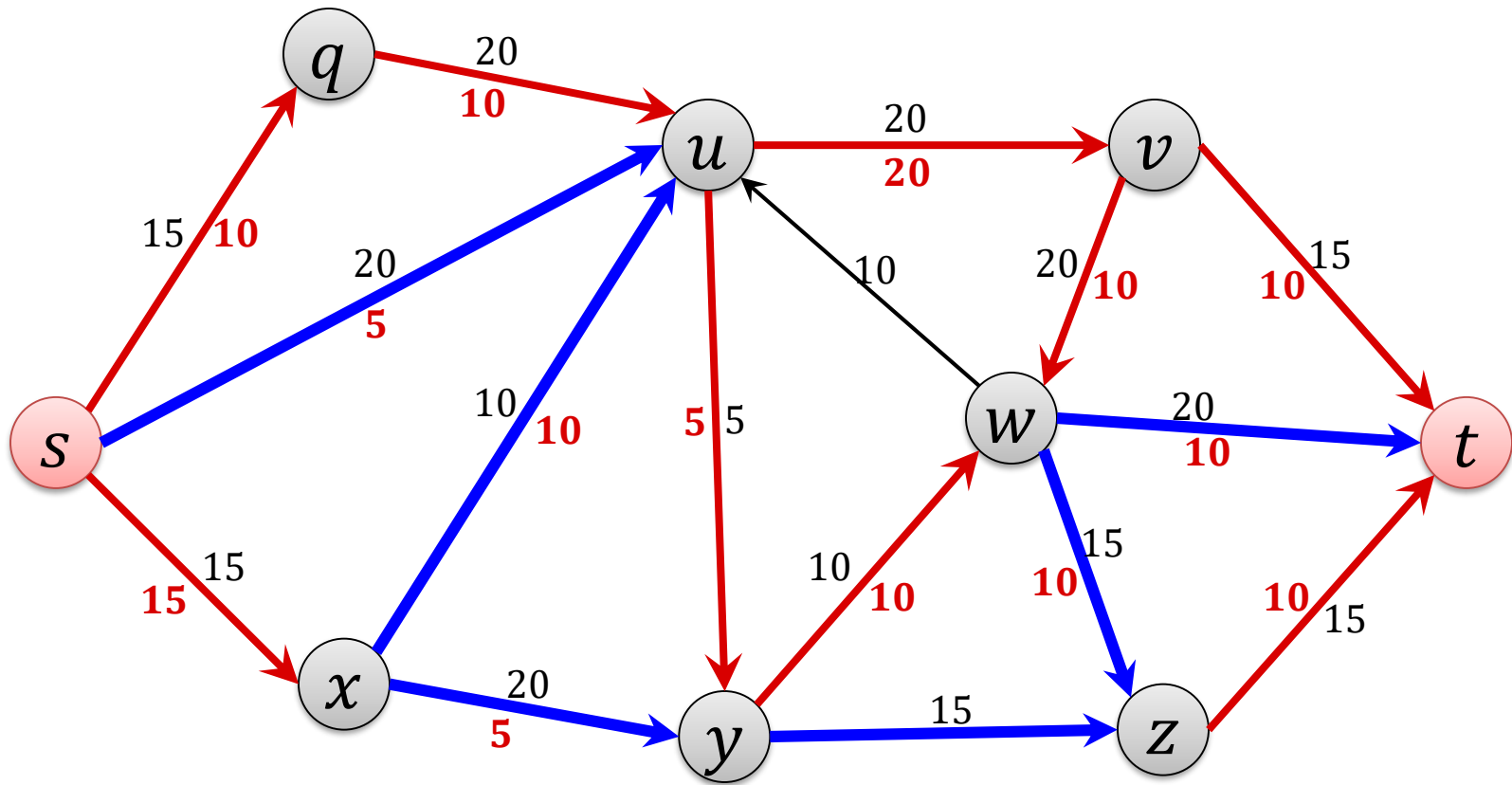
# Augmenting Path

## Residual Graph $G_f$



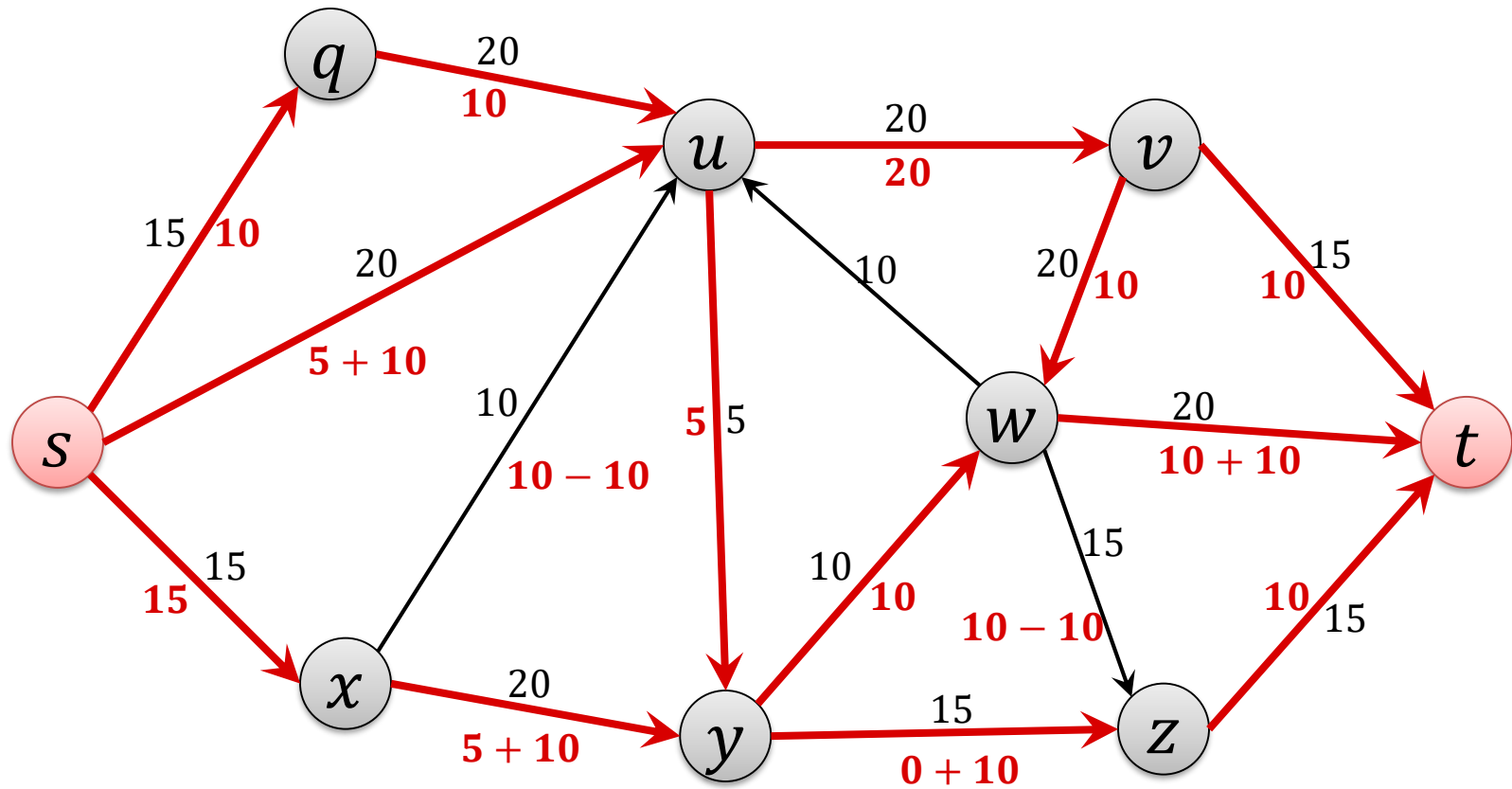
# Augmenting Path

## 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**  $> 0$ .

**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  $P$ :

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

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

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

# Augmented Flow

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

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

**Proof:**

# 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:**

# 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.      $f' := \text{augment}(f, P)$ ;
  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

$$C = \sum_{e \text{ out of } s} c_e.$$

**Proof:**

# Ford-Fulkerson Running Time



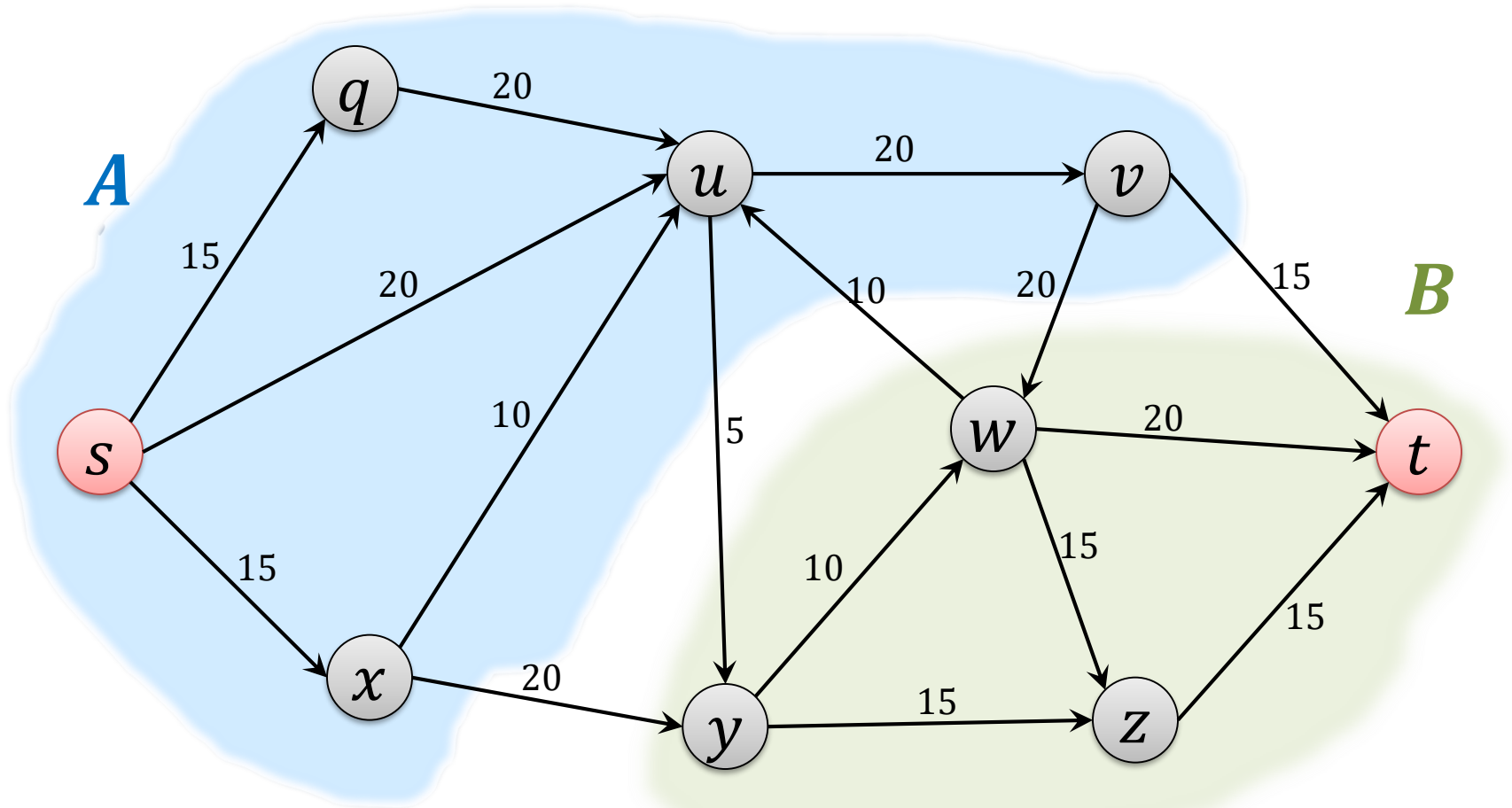
**Theorem:** If all edge capacities are integers, the Ford-Fulkerson algorithm can be implemented to run in  $O(mC)$  time.

**Proof:**

# $s$ - $t$ Cuts

## Definition:

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

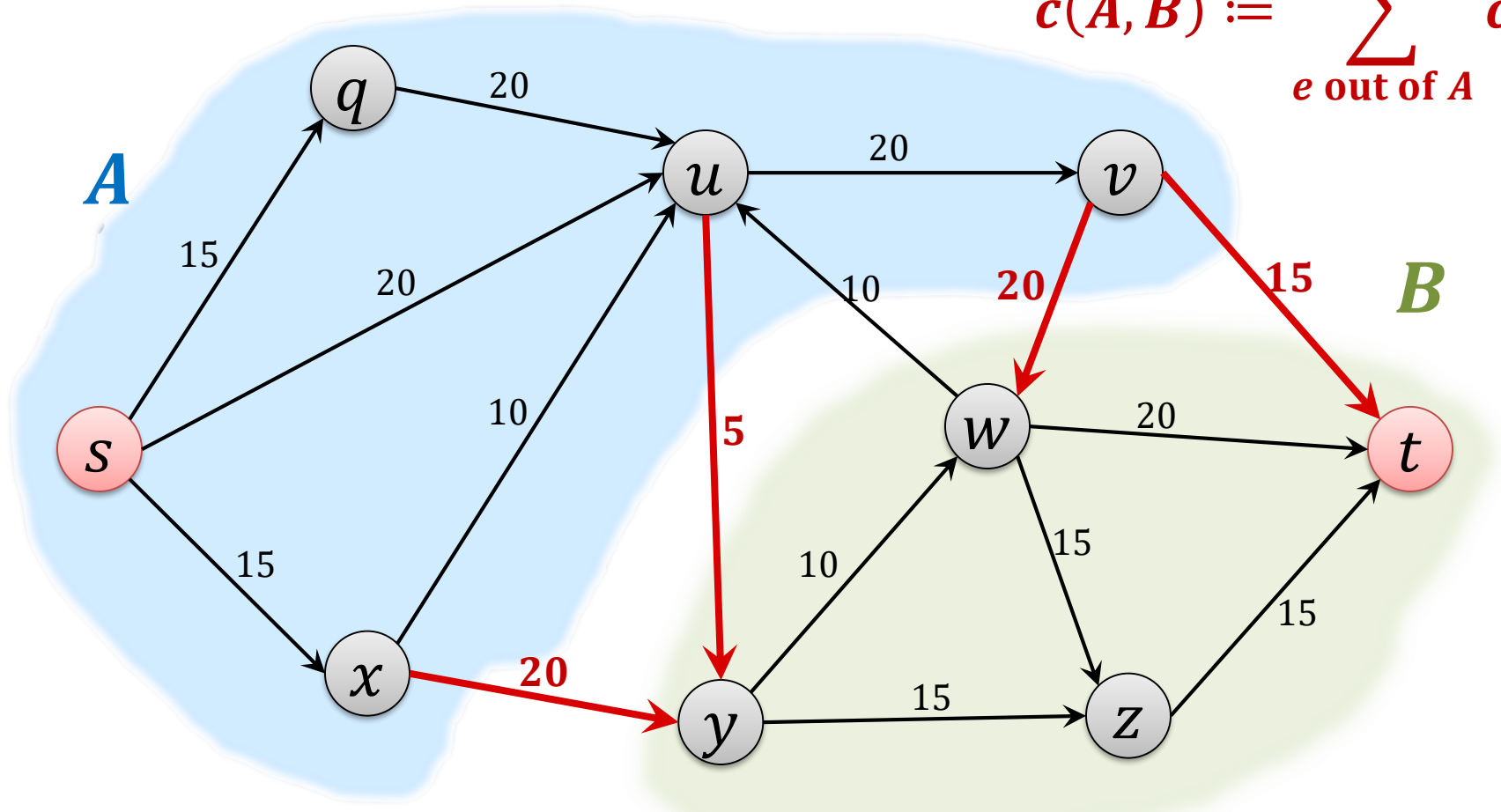


# Cut Capacity

## 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,

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

**Proof:**

# 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:**

# Upper Bound on Flow Value

## Lemma:

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

## Proof:

# 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

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

**Proof:**

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

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

**Proof:**

# 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

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

**Proof:**

# Ford-Fulkerson Gives Optimal Solution



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

**Proof:**

# Min-Cut Algorithm

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:**

# 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:

## 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.

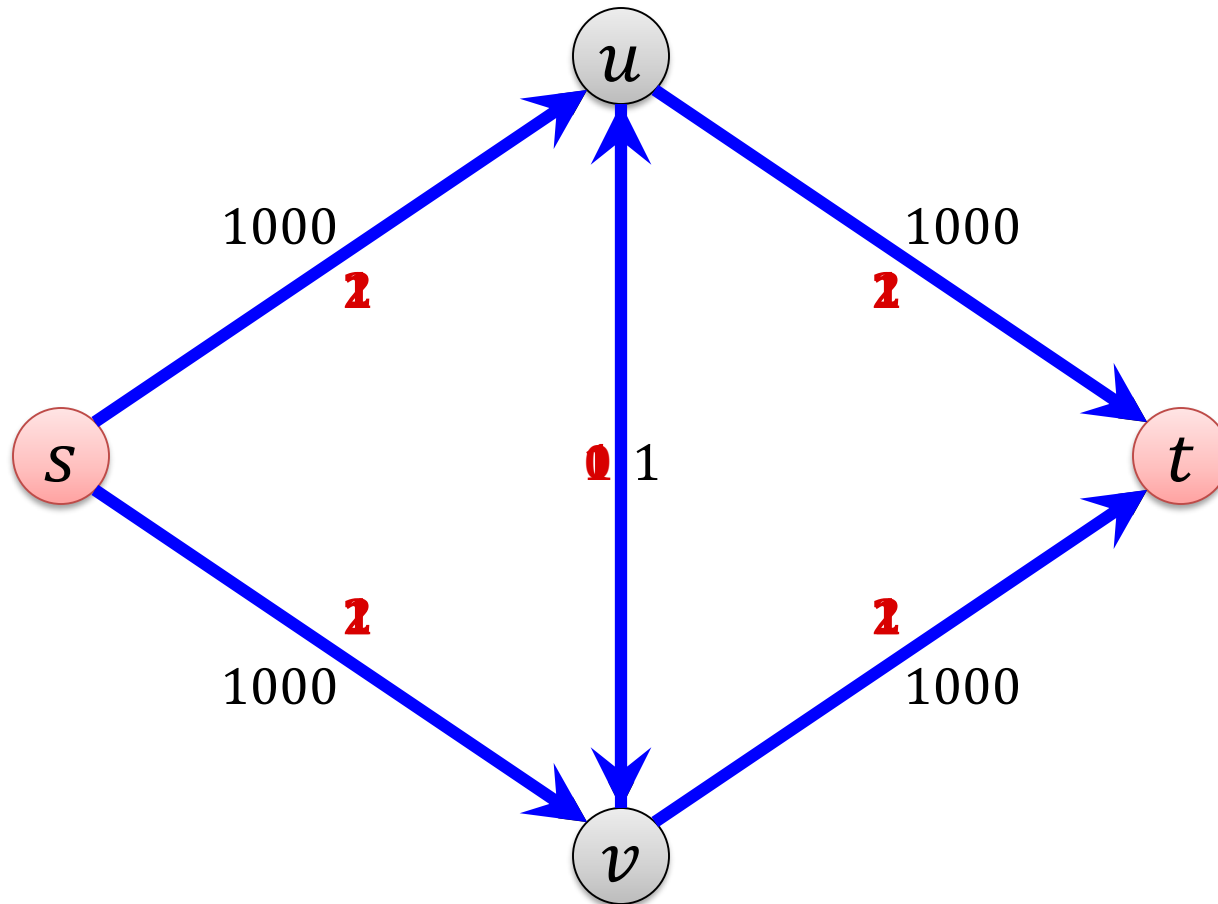
**Proof:**

# Non-Integer Capacities

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
- **Scaling parameter  $\Delta$ :**  
(initially,  $\Delta = \lceil \max c_e \rceil$  rounded down to next power of 2")
- As long as there is an augmenting path that improves the flow by at least  $\Delta$ , 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  $\leq 1 + \lfloor \log_2 C \rfloor$ .

- **$\Delta$ -scaling phase:** Time during which scaling parameter is  $\Delta$

# Length of a Scaling Phase



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

# Length of a Scaling Phase



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

# Running Time: Scaling Max Flow Alg.



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

# Strongly Polynomial Algorithm

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

$$O(mC)$$

- Time of algorithm with scaling parameter:

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

- $O(\log C)$  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

$$O(m^2 n)$$

# 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 last year's 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  $m \geq n^{1+\epsilon}$  [King,Rao,Tarjan 1992/1994]  
(for every constant  $\epsilon > 0$ )
  - For sparse graphs with  $m \leq n^{16/15-\delta}$  [Orlin, 2013]

# Maximum Flow Applications

- Maximum flow has many applications
- Reducing a problem to a max flow problem can even be seen as an important algorithmic technique
- Examples:
  - related network flow problems
  - computation of small cuts
  - computation of matchings
  - computing disjoint paths
  - scheduling problems
  - assignment problems with some side constraints
  - ...