Algorithmen und Datenstrukturen

11. GRAPHEN 3
FLÜSSE UND SPANNBÄUME

WeST

Algorithmen und Datenstrukturen

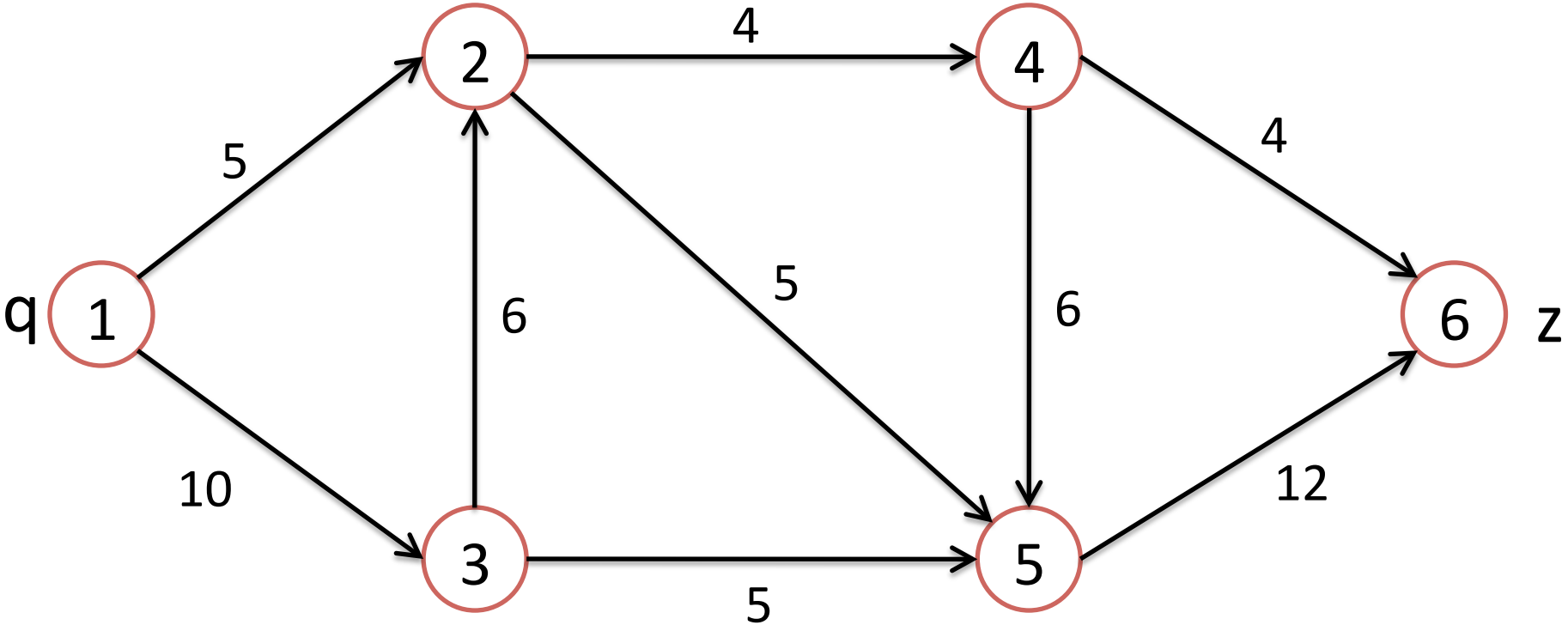
11.1. BERECHNUNG MAXIMALER FLÜSSE

Maximaler Fluss

- Logistische Aufgabe:
  - Verteilungsnetz mit Kapazitäten
    - Wasserrohre
    - Förderbänder
    - Paketvermittlung im Rechnernetz
- Quelle liefert (beliebig viele) Objekte pro Zeiteinheit
- Senke verbraucht diese
- Jede Verbindung hat eine maximale Kapazität c und einen aktuellen Fluss f
- Wie hoch ist die Übertragungskapazität?

Problemformalisierung

Gegeben:
Digraph (V,E) mit Kapazitätsfunktion c: E → $\mathbb{R}$,
Knoten q∈V (Quelle) und z∈V (Ziel)



Definition: Fluss

Ein Fluss f von q∈V nach z∈V ist eine Funktion

$$f_{q,z} : E \to \mathbb{R}$$

für die die folgenden zwei Bedingungen gelten:

1. Die Kapazitäten werden eingehalten:

$$\forall e \in E : f_{q,z}(e) \leq c(e)$$

2. Was in einen Knoten hereinfließt, muss wieder herausfließen (mit Ausnahme von q und z):

$$\forall v \in V \setminus \{q, z\} : \sum_{u \in P(v)} f((u,v)) = \sum_{w \in S(v)} f((v,w))$$

wobei $P(v) = \{u | (u,v) \in E\}$ (Vorgänger von v)
und $S(v) = \{w | (v,w) \in E\}$ (Nachfolger von v)

Maximaler Fluss

- Wert eines Flusses

$$val(G, f_{q,z}) = \sum_{u \in S(q)} f_{q,z}((q,u))$$

- Gesucht: Wert eines maximalen Flusses

$$\max_f \{val(G,f) | f \text{ ist Fluss von q nach z}\}$$

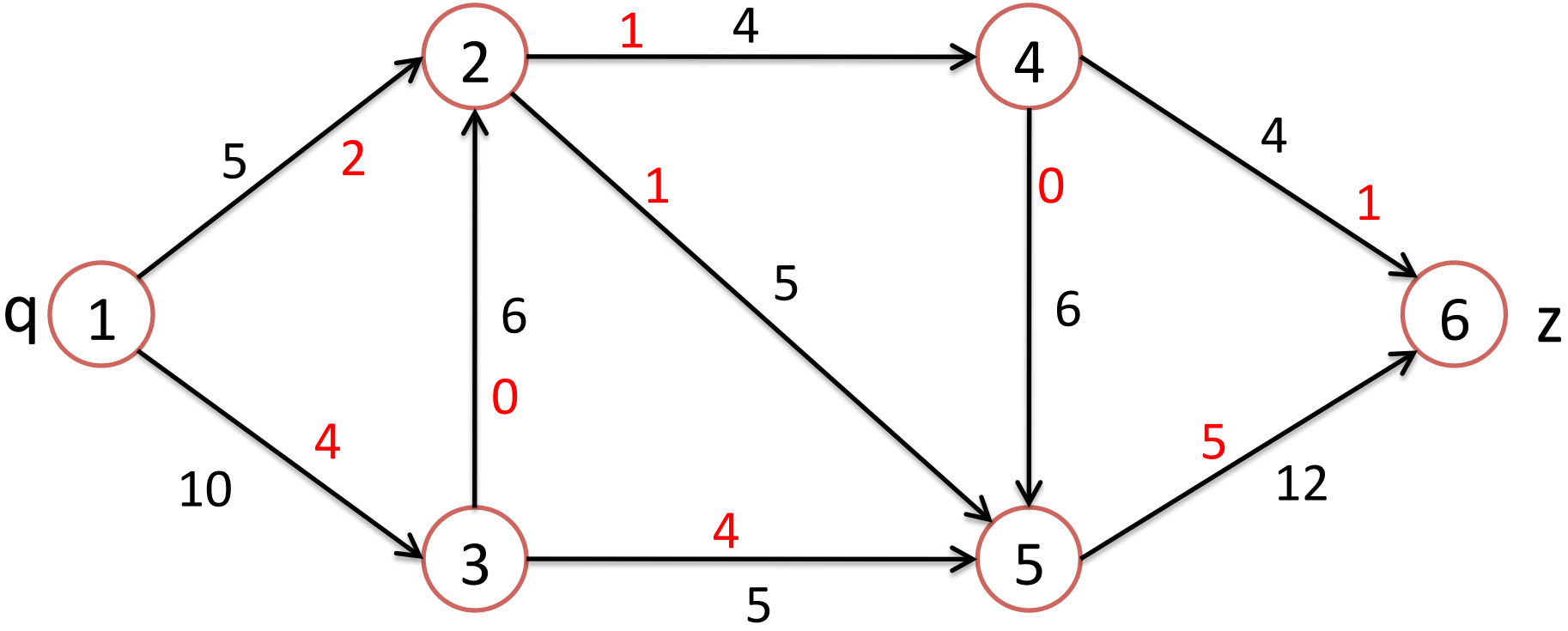
Beispiel 1/3

Definiere f_1 durch

$f_1((1,2))=2$ $f_1((1,3))=4$ $f_1((2,4))=1$ $f_1((2,5))=1$ $f_1((3,2))=0$

$f_1((3,5))=4$ $f_1((4,5))=0$ $f_1((4,6))=1$ $f_1((5,6))=5$

$\rightarrow$ val(G,f_1)=6



WeST

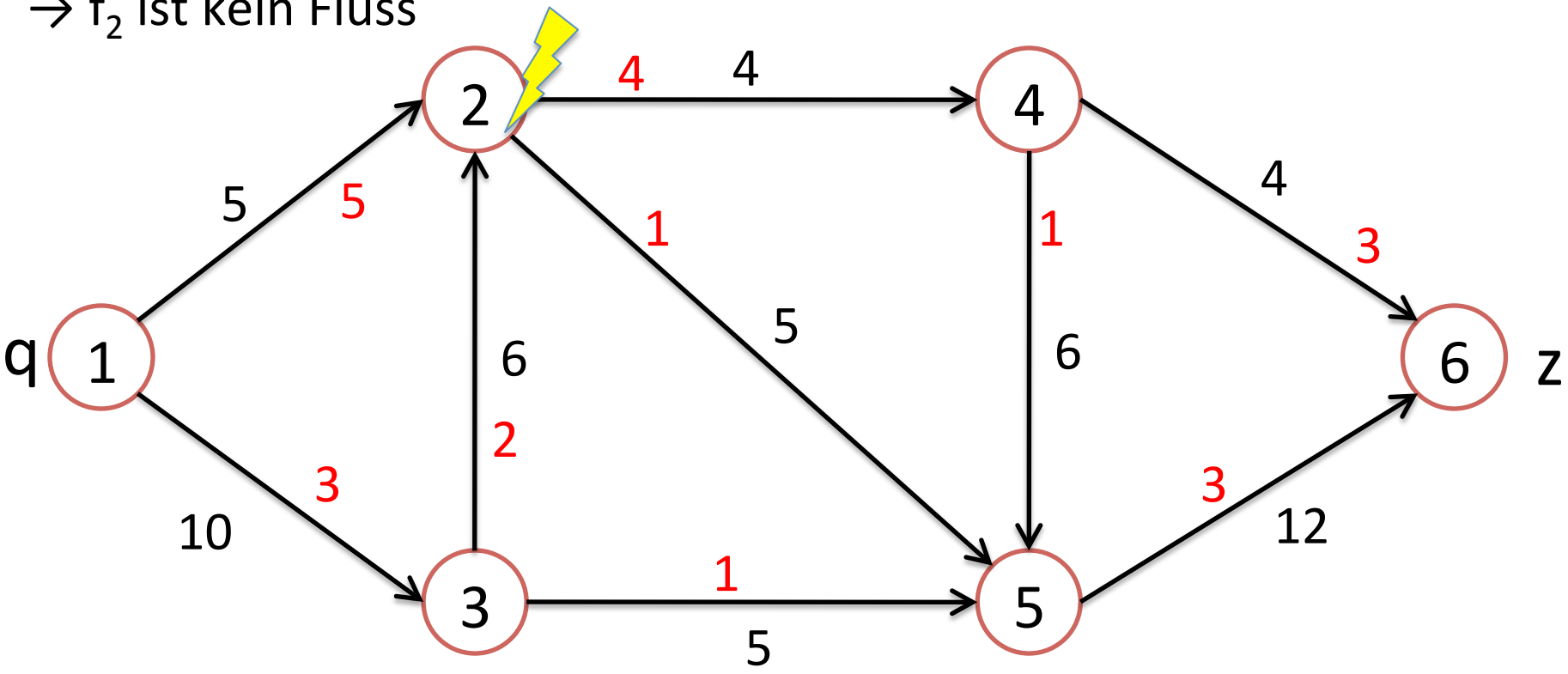
Beispiel 2/3

Definiere f_2 durch

$f_2((1,2))=5$ $f_2((1,3))=3$ $f_2((2,4))=4$ $f_2((2,5))=1$ $f_2((3,2))=2$

$f_2((3,5))=1$ $f_2((4,5))=1$ $f_2((4,6))=3$ $f_2((5,6))=3$

$\rightarrow f_2$ ist kein Fluss



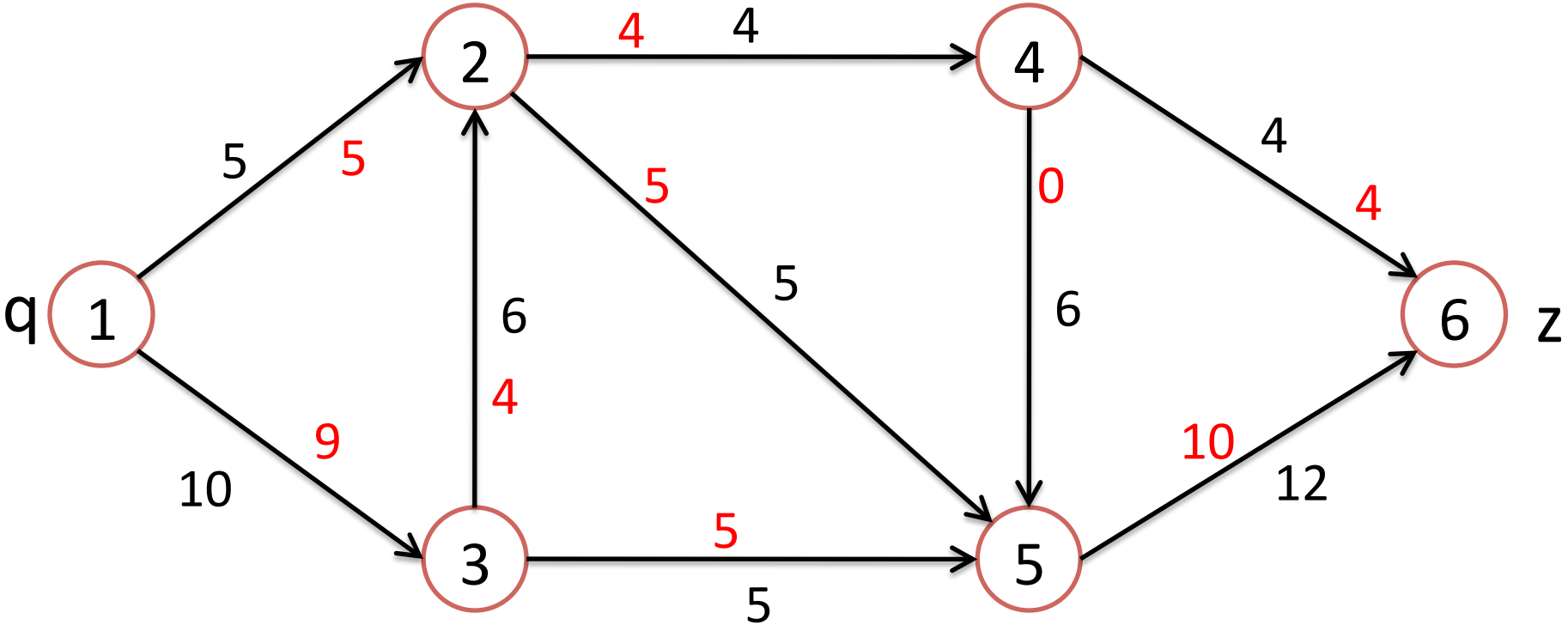
Beispiel 3/3

Definiere f_3 durch

$f_3((1,2))=5$ $f_3((1,3))=9$ $f_3((2,4))=4$ $f_3((2,5))=5$ $f_3((3,2))=4$

$f_3((3,5))=5$ $f_3((4,5))=0$ $f_3((4,6))=4$ $f_3((5,6))=10$

$\rightarrow$ val(G,f_3)=14 (f_3 ist maximal)



Berechnung eines maximalen Flusses

- Ford-Fulkerson-Algorithmus
  - Effiziente Bestimmung eines maximalen Flusses von q nach z
  - Vorgehensweise: Mischung aus Greedy und Zufallsauswahl
- Prinzip: *Füge so lange verfügbare Pfade zum Gesamtfluss hinzu wie möglich.*

- Finden eines nutzbaren Pfades etwa durch Tiefensuche
- Für Kanten werden drei Werte notiert:
  1. Aktueller Fluss f entlang der Kante
     - Im initialisierten Graphen ist dieser Wert überall 0
  2. Vorgegebene Kapazität c
  3. Abgeleitete noch verfügbare (Rest-)Kapazität c-f

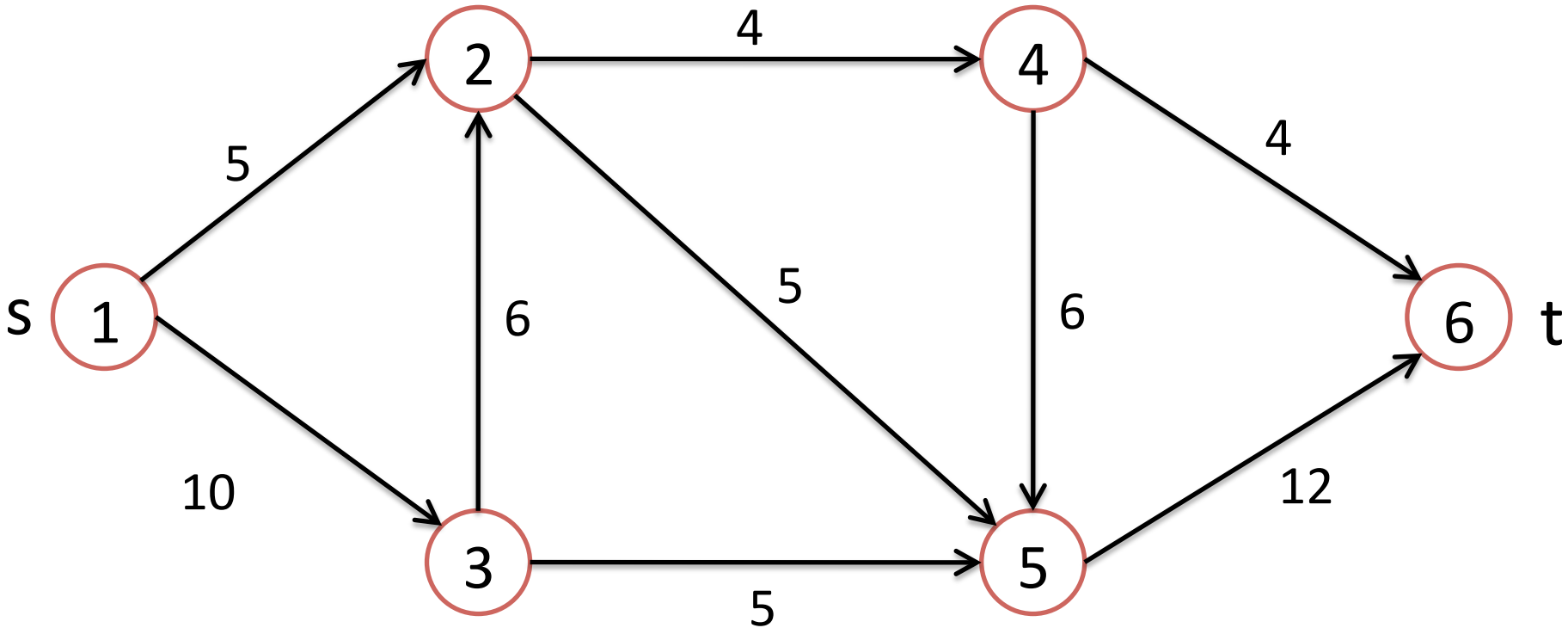
Ford-Fulkerson-Algorithmus (Pseudocode)

Vorläufige Version:

```
initialisiere Graph mit leerem Fluss;
do
   wähle nutzbaren Pfad aus;
   füge Fluss des Pfades zum Gesamt-
       fluss hinzu;
while noch nutzbarer Pfad verfügbar
```

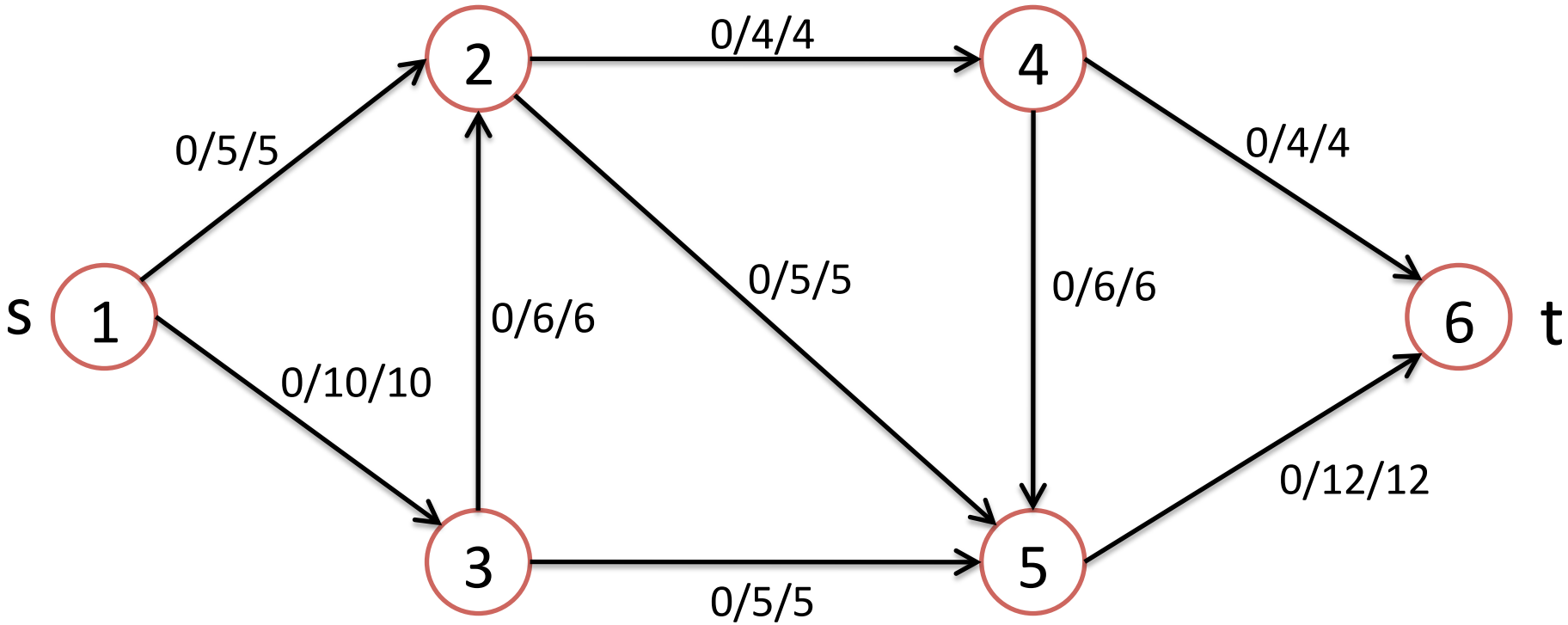
- Nutzbarer Pfad
  - (zyklenfreier) Pfad von Quelle q zum Ziel z, der an allen Kanten eine verfügbare Kapazität hat
- Nutzbarer Fluss
  - Minimum der verfügbaren Kapazitäten der einzelnen Kanten

Beispiel: Ford-Fulkerson-Algorithmus



Graph mit Kapazitäten

Beispiel: Ford-Fulkerson-Algorithmus

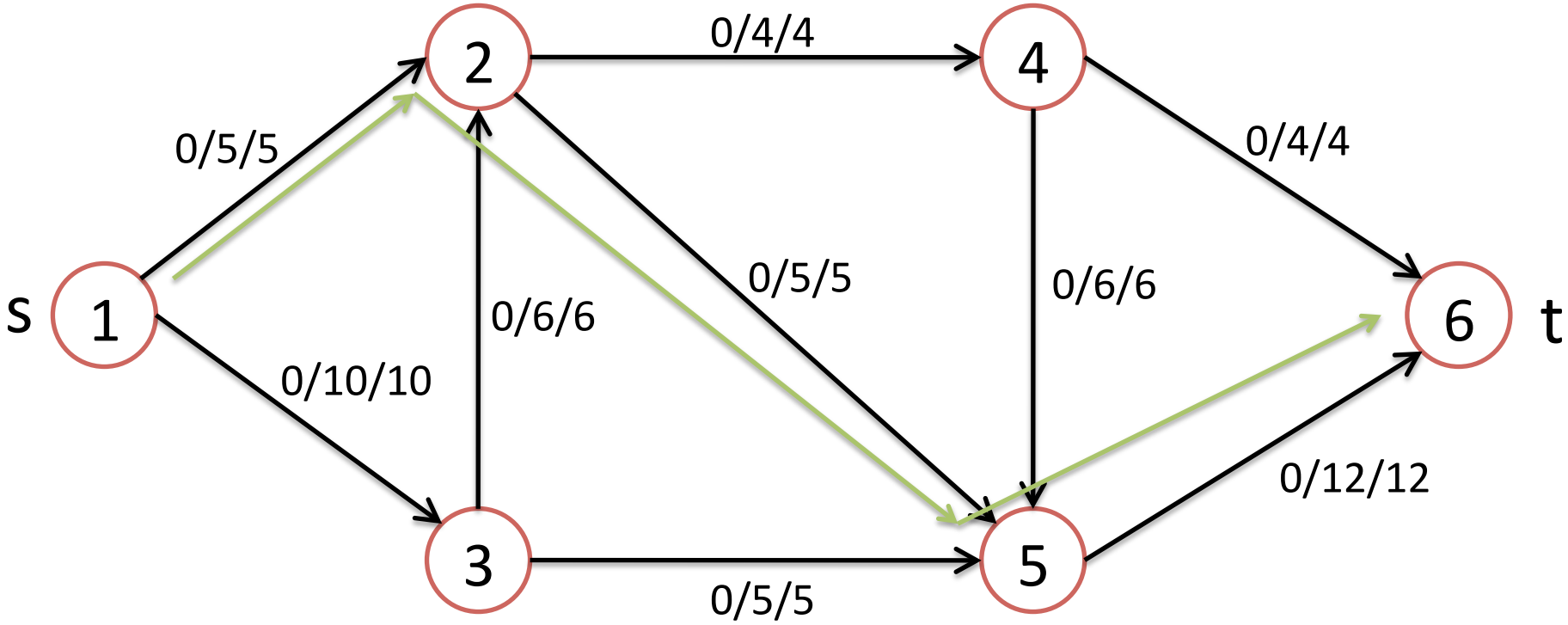


Initialisiere mit Fluss: 0

Notation: <aktueller Fluss f> / <Kapazität c> / <verfügbare Kapazität c-f>

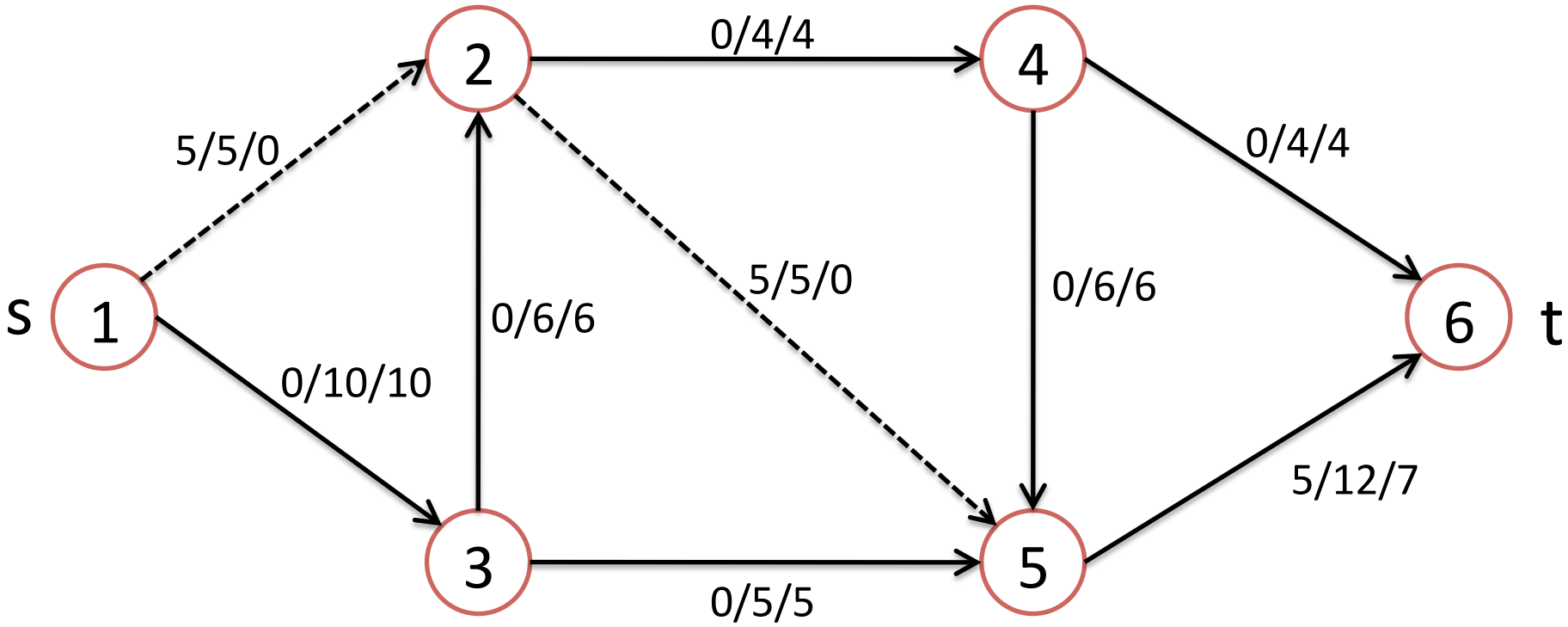
Beispiel: Ford-Fulkerson-Algorithmus

Auswahl nutzbarer Pfade: zufällig oder geeignete Heuristik
Anmerkung: es gibt kürzere Pfade und mit höherer Kapazität



Auswahl eines Pfades: 1 → 2 → 5 → 6 Nutzbarer Fluss: 5

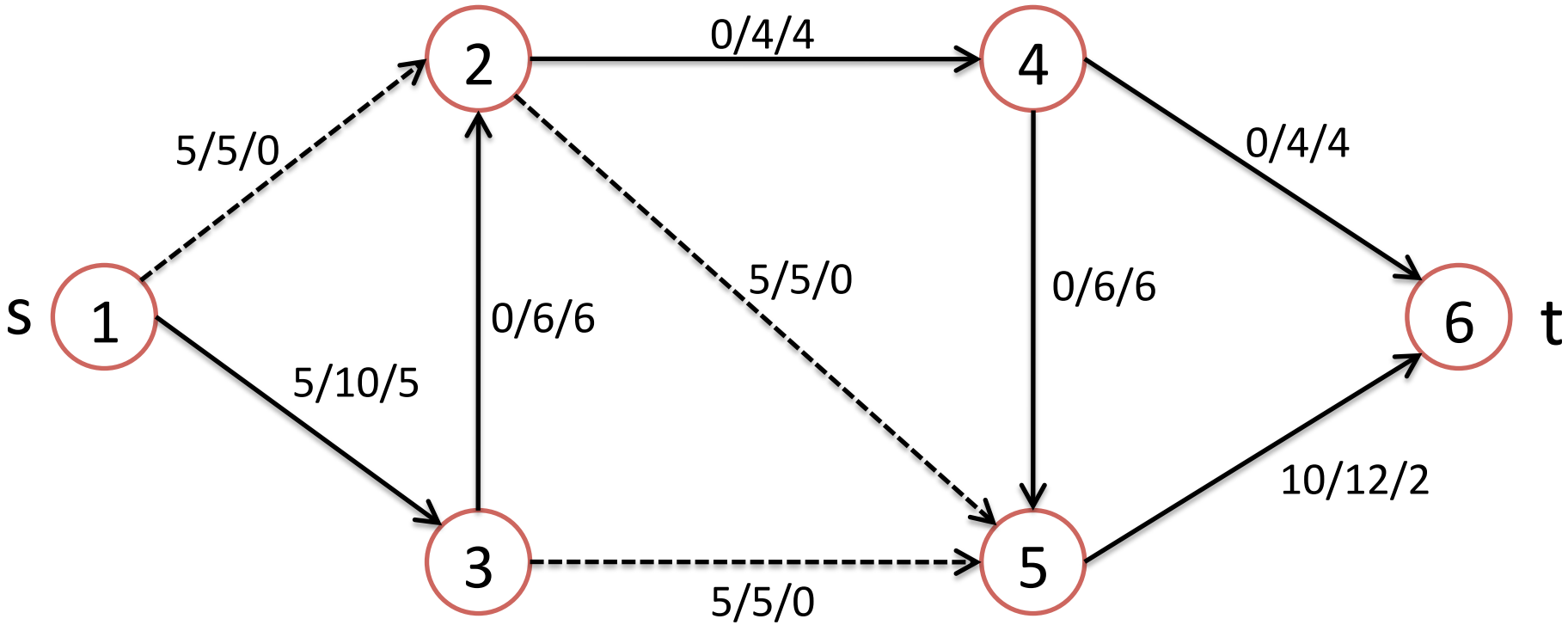
Beispiel: Ford-Fulkerson-Algorithmus



Aktualisieren des Flusses

Auswahl eines Pfades: 1 → 3 → 5 → 6 Nutzbarer Fluss: 5

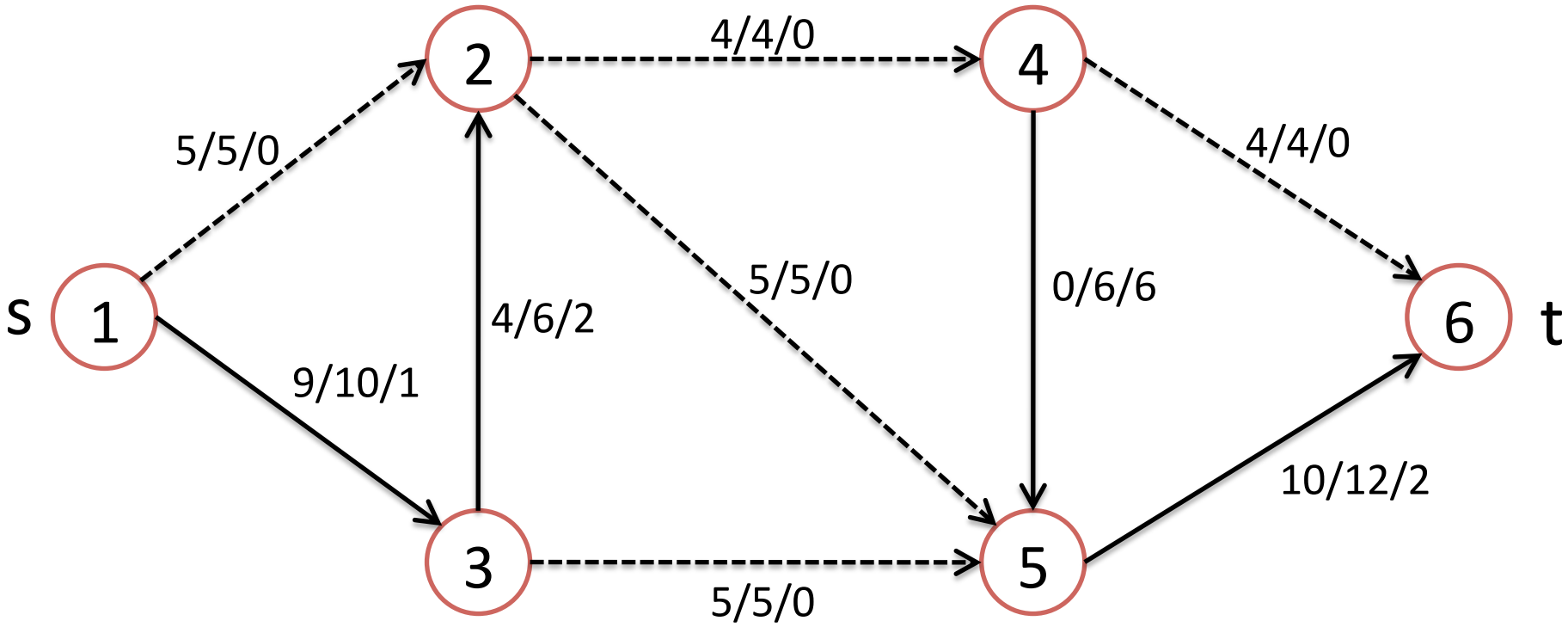
Beispiel: Ford-Fulkerson-Algorithmus



Aktualisieren des Flusses

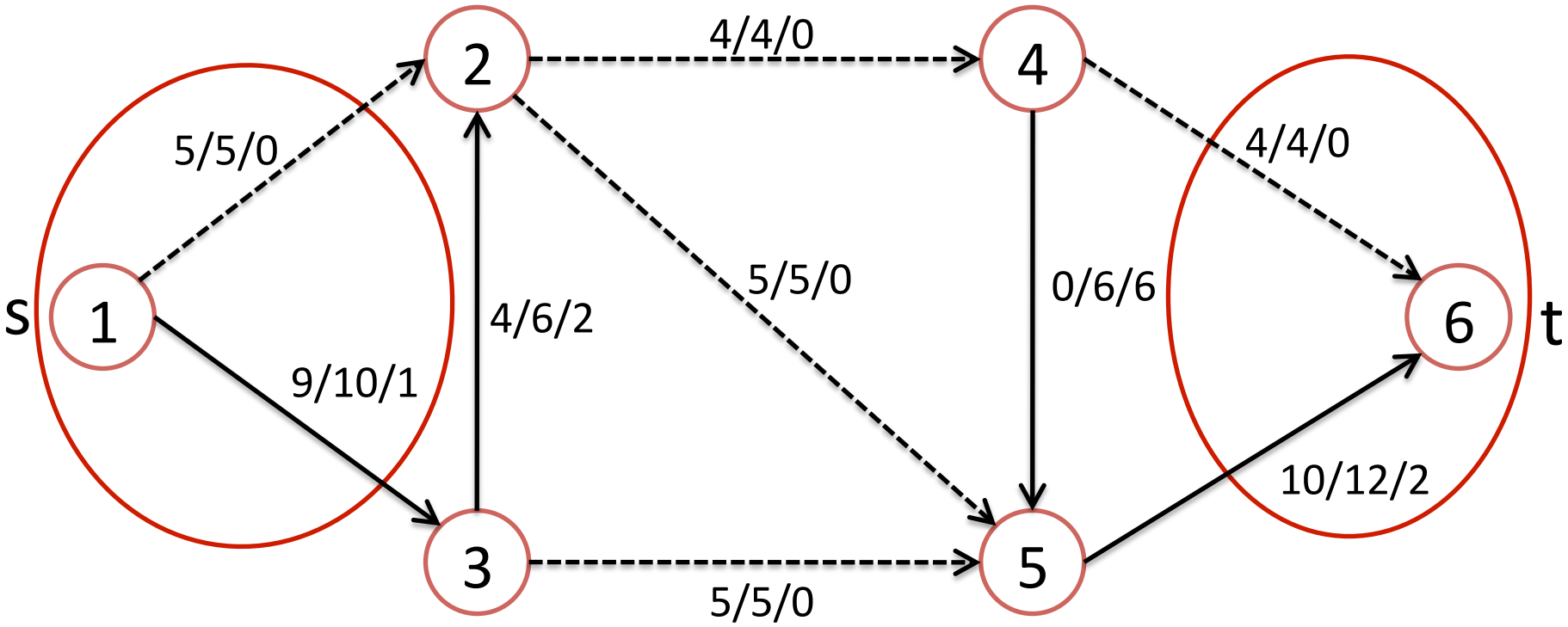
Auswahl eines Pfades: 1 → 3 → 2 → 4 → 6 Nutzbarer Fluss: 4

Beispiel: Ford-Fulkerson-Algorithmus



Aktualisieren des Flusses

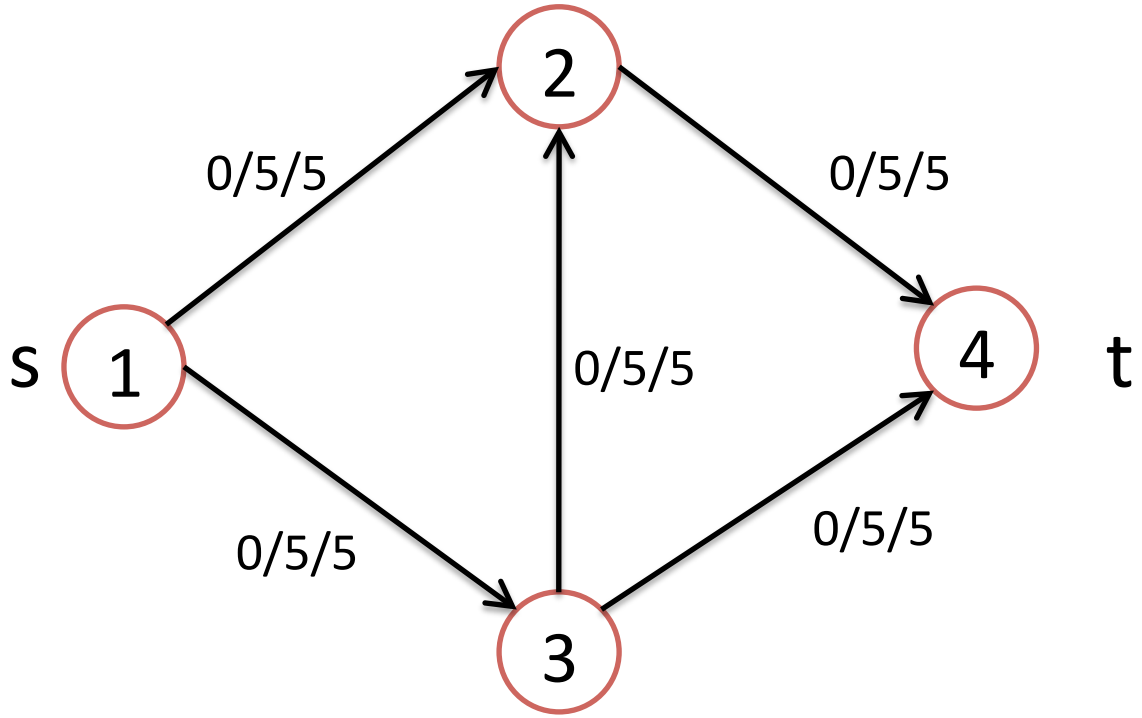
Beispiel: Ford-Fulkerson-Algorithmus



Keine weiteren Pfade möglich ➔ Berechnung beendet
Maximaler Fluss: 14

Problem: Ungünstige Pfadwahl

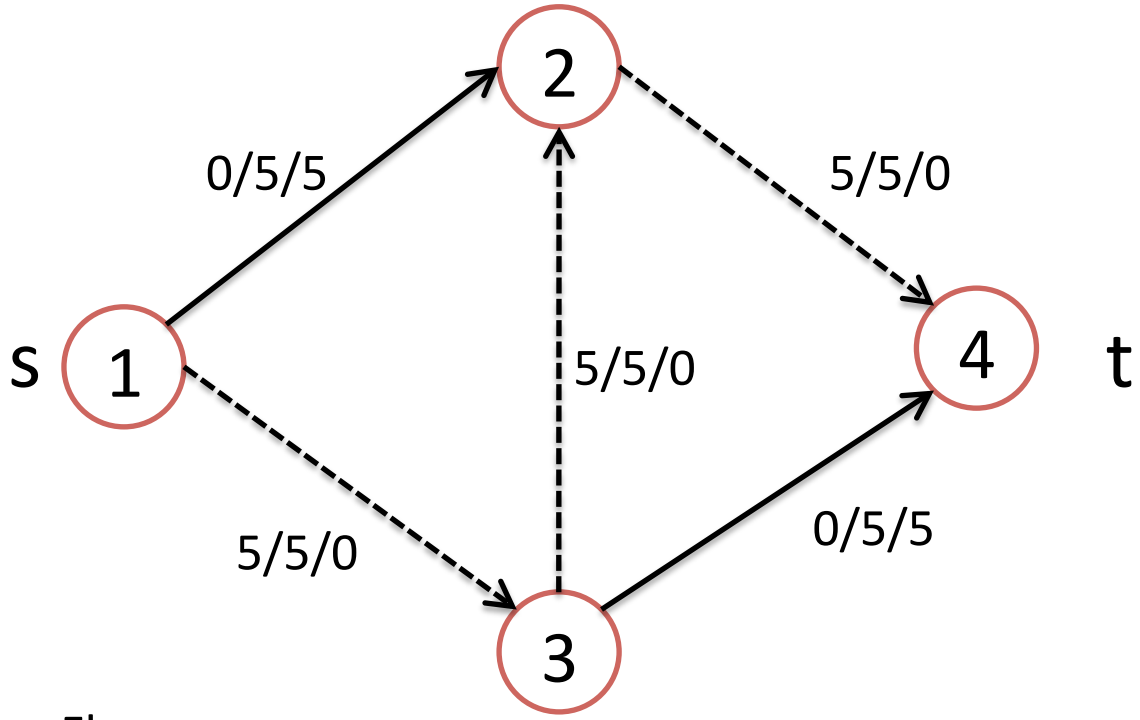
- Die bisher betrachtete Version des Algorithmus ist nicht immer optimal



Auswahl eines Pfades: 1 → 3 → 2 → 4 Nutzbarer Fluss: 5

Problem: Ungünstige Pfadwahl

- Die bisher betrachtete Version des Algorithmus ist nicht immer optimal

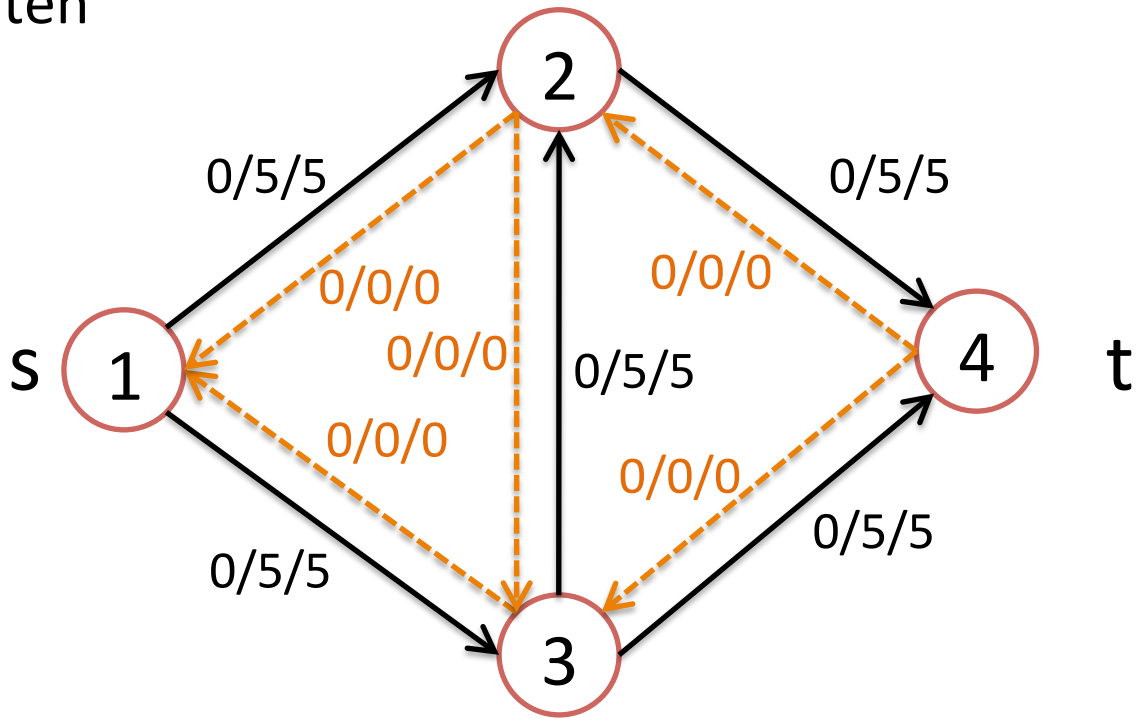


Aktualisieren des Flusses

Keine weitere Pfadwahl mehr möglich (optimal Lösung aber Pfade (1,2,4) und (1,3,4))

Problem: Ungünstige Pfadwahl

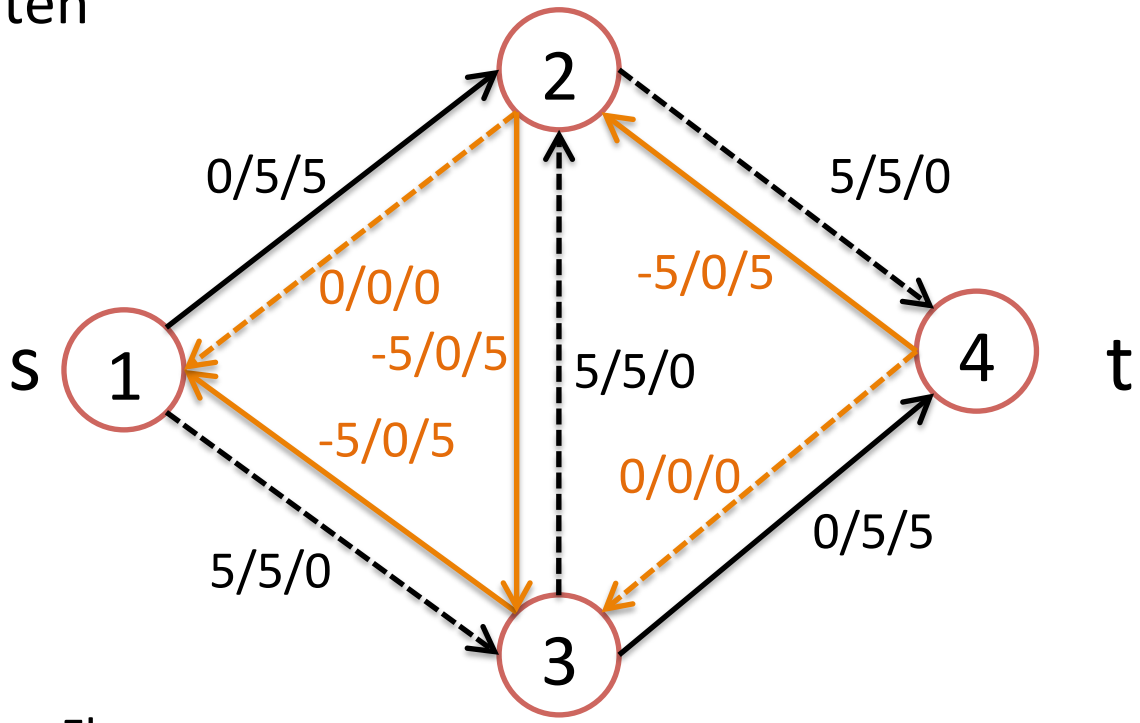
- Problem: Fluss kann nicht zurückgenommen werden
- Lösung: erlaube entgegengesetzte Flussrichtung durch Rückkanten



Auswahl eines Pfades: 1 → 3 → 2 → 4 Nutzbarer Fluss: 5

Problem: Ungünstige Pfadwahl

- Problem: Fluss kann nicht zurückgenommen werden
- Lösung: erlaube entgegengesetzte Flussrichtung durch Rückkanten

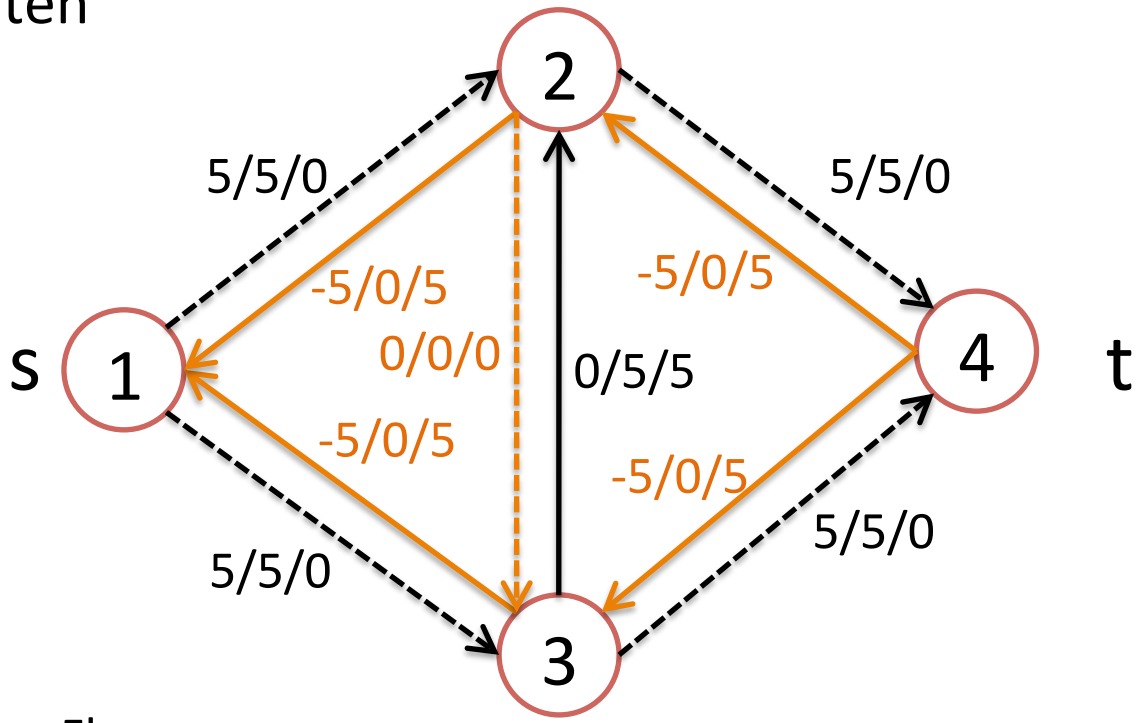


Aktualisieren des Flusses

Auswahl eines Pfades: 1 → 2 → 3 → 4 Nutzbarer Fluss: 5

Problem: Ungünstige Pfadwahl

- Problem: Fluss kann nicht zurückgenommen werden
- Lösung: erlaube entgegengesetzte Flussrichtung durch Rückkanten



Aktualisieren des Flusses

Keine weiteren Pfade möglich ➔ Berechnung beendet (Maximaler Fluss: 10)

Ford-Fulkerson-Algorithmus (Pseudocode)

Finale Version:

```
für jede Kante (u,v) füge Kante (v,u)
   mit Kapazität 0 ein
initialisiere Graph mit leerem Fluss;
do
   wähle nutzbaren Pfad aus;
   füge Fluss des Pfades zum Gesamt-
       fluss hinzu;
while noch nutzbarer Pfad verfügbar
```

Analyse

Theorem (Terminierung)

Sind alle Kapazitäten in G nicht-negativ und *rational*, dann terminiert der Algorithmus von Ford-Fulkerson nach endlicher Zeit.

Ohne Beweis

Theorem (Laufzeit)

Ist X der Wert eines maximales Flusses in G=(V,E) und sind alle Kapazitäten in G nicht-negativ und *ganzzahlig*, so hat der Algorithmus von Ford-Fulkerson eine Laufzeit von O(|E|X).

Ohne Beweis

Analyse

Theorem (Korrektheit)

Sind alle Kapazitäten in G nicht-negativ und *rational*, dann berechnet der Algorithmus von Ford-Fulkerson den Wert eines maximalen Flusses.

Ohne Beweis

Ford-Fulkerson-Algorithmus

- Wahl des Pfades beeinflusst Anzahl benötigter Iterationen

- Verfahren von Edmonds und Karp
  - Anzahl der Pfade die in einem Graphen G = (V,E) bis zum Finden des maximalen Flusses verfolgt werden müssen ist kleiner als $|V||E|$, wenn jeweils der kürzeste Pfad von Quelle q zu Ziel z gewählt wird
  - Daher: Auswahl des nächsten (kürzesten) Pfades kann basierend auf einer Variante der Breitensuche erfolgen
  - Verbessert die Laufzeit auf $O(|V||E|^2)$

Algorithmen und Datenstrukturen

11.1. BERECHNUNG MAXIMALER FLÜSSE

ZUSAMMENFASSUNG

Zusammenfassung

- Berechnung eines maximales Flusses
- Algorithmus von Ford-Fulkerson
  - *Idee:* Füge so lange verfügbare Pfade zum Gesamtfluss hinzu wie möglich
  - Einfügung von Rückwärtskanten notwendig um Optimalität zu garantieren
  - Laufzeit abhängig von Ergebniswert
  - Nur für nicht-negative und rationale Kapazitäten korrekt

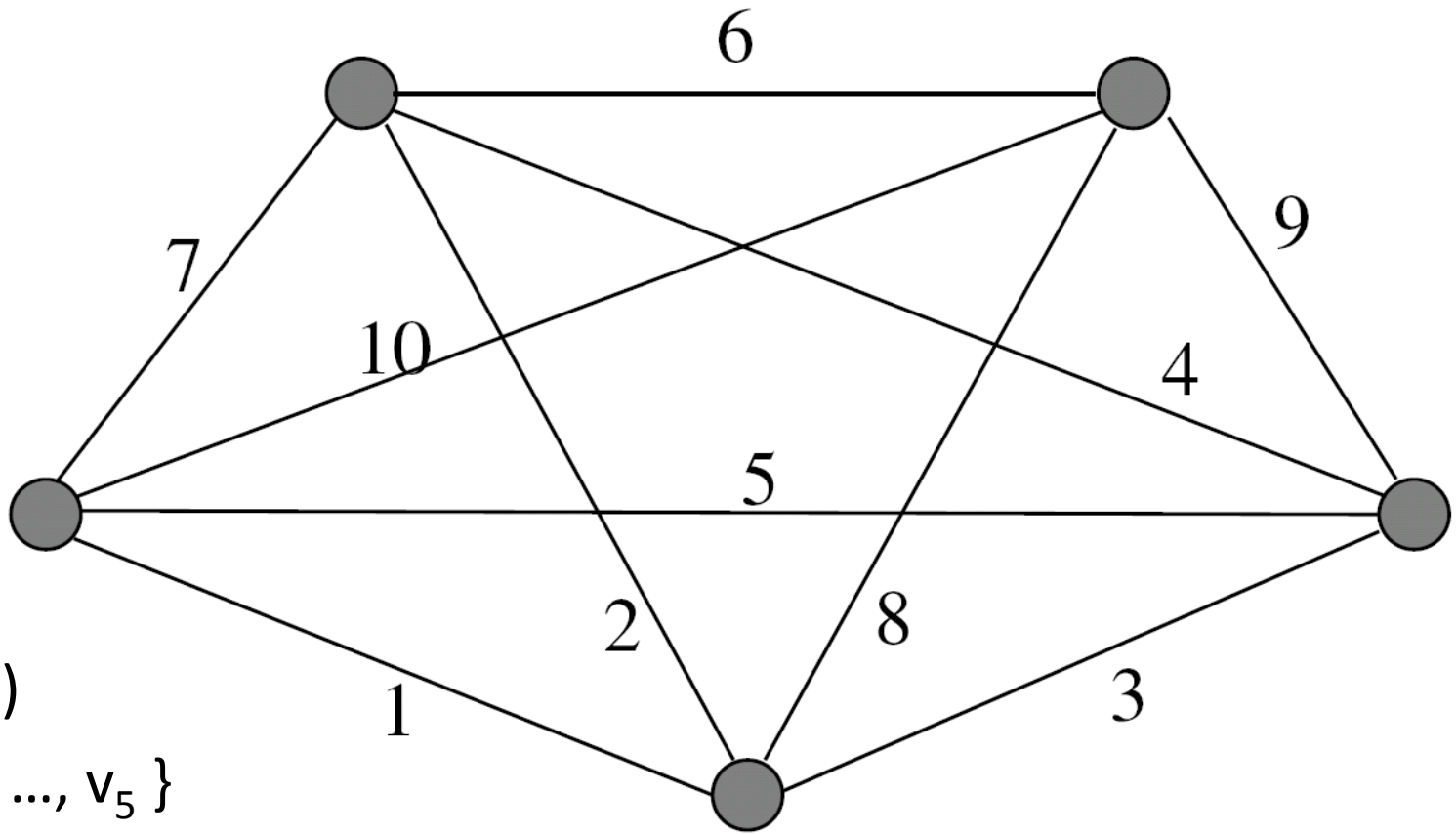
Algorithmen und Datenstrukturen

11.2. SPANNBÄUME

Beispiel: Kommunikationsnetz

- Zwischen n Knotenpunkten $v_1, \ldots, v_n$ soll ein möglichst günstiges Kommunikationsnetz geschaltet werden, so dass jeder Knotenpunkt mit jedem anderen verbunden ist, ggf. auf einem Umweg über andere Knotenpunkte.
- Bekannt sind Kosten $c_{i,j}$ für die direkte Verbindung zwischen v_i und v_j, $1 \leq i,j \leq n$.
- Alle Kosten $c_{i,j}$ seien verschieden und größer als Null.
- Modellierung als gewichtete, ungerichtete und vollständige Graphen mit Gewichtsfunktion c

Beispiel: Eingabe für Kommunikationsnetz



- G=(V,E)
- V={ v_1, …, v_5 }
- E={ $(v_1,v_2),(v_1,v_3),(v_1,v_4),$
  $(v_1,v_5),(v_2,v_3),(v_2,v_4),(v_2,v_5),(v_3,v_4),(v_3,v_5),(v_4,v_5)$ }
- $c((v_1,v_2))=6$, $c((v_1,v_3))=7$, etc.; abgekürzt: $c_{1,2}=6$, $c_{1,3}=7$, etc.

Problemstellung: Finde minimal aufspannenden Baum

Einige Definitionen (Wdh.):

- Ein Graph G=(V,E) heisst *zusammenhängend*, wenn für alle $v,w \in V$ ein Weg von v nach w in G existiert
- Ein Graph G=(V,E) enthält einen Kreis, wenn es unterschiedliche Knoten $v_1,\ldots,v_n \in V$ gibt, so dass
  $\{v_1,v_2\},\ldots,\{v_{n-1},v_n\},\{v_n,v_1\} \in E$
- Ein Graph G=(V,E) heisst *Baum*, wenn er
  - zusammenhängend ist und
  - keinen Kreis enthält

Problemstellung: Finde minimal aufspannenden Baum

Einige Definitionen (Wdh.):

- Ein Graph G'=(V',E') heisst *Teilgraph* von G=(V,E), wenn V' ⊆ V und E'⊆ E ∩ (V'xV')
- Ein Graph G'=(V',E') heisst *induzierter Teilgraph* von G=(V,E) bzgl V' ⊆ V, wenn E'= E ∩ (V'xV')
- Ein Graph G'=(V',E') heisst *Spannbaum* von G=(V,E), wenn
  - G' ist Teilgraph von G,
  - V'=V und
  - G' ist ein Baum

Frage: Wie viele Kanten hat ein Spannbaum?

Problemstellung: Finde minimal aufspannenden Baum

Weitere Definitionen:

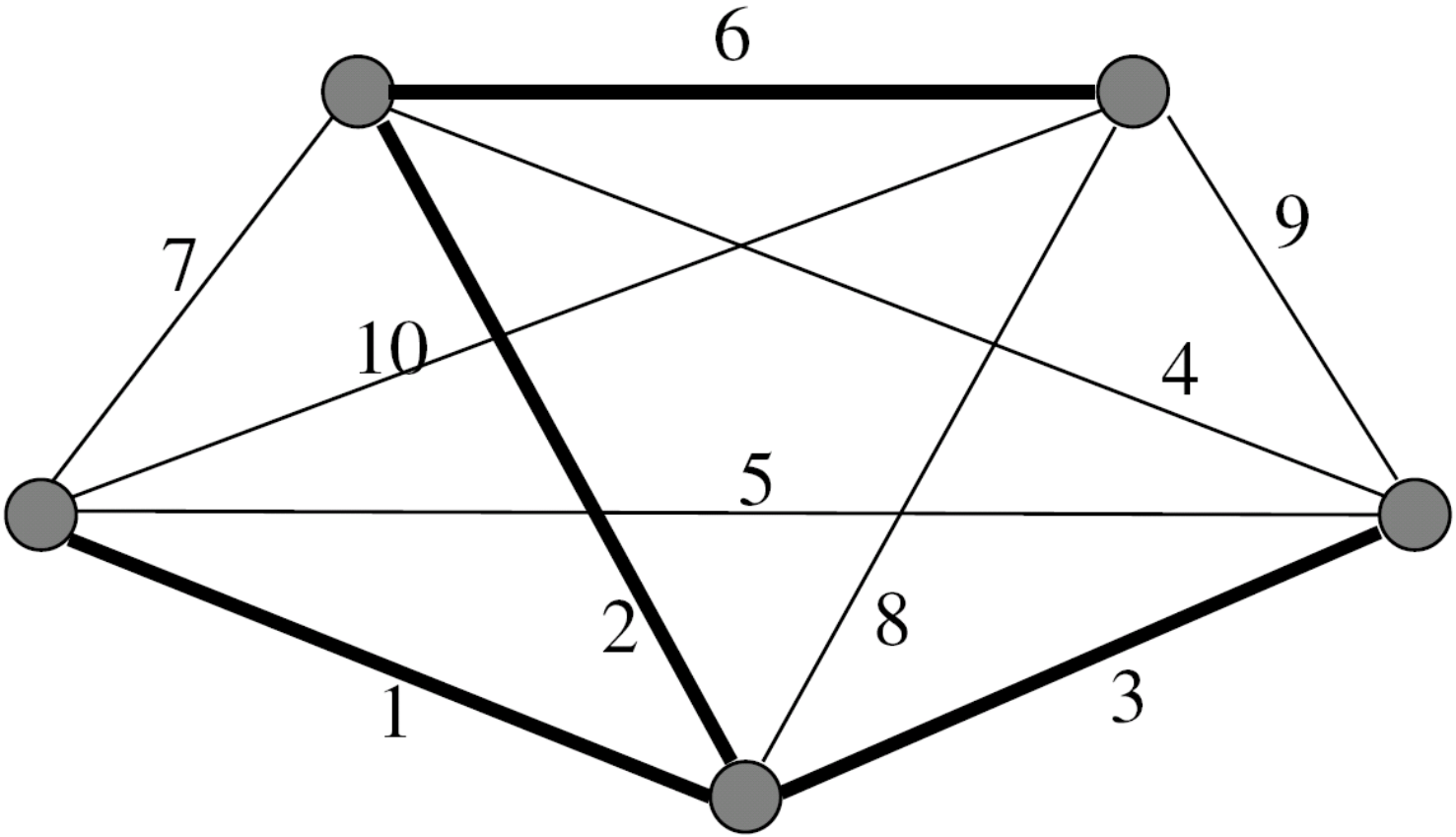
- Das *Gewicht* eines Graphen G=(V,E) ist

$$C(G) = \sum_{(i,j) \in E} c_{i,j}$$

- Ein Graph G‘=(V‘,E‘) ist ein *minimaler Spannbaum* von G=(V,E) wenn
  - G‘ ist ein Spannbaum von G
  - Unter allen Spannbäumen von G hat G‘ minimales Gewicht

Aufgabe: Gegeben G=(V,E), finde einen minimalen Spannbaum von G

Beispiel



Lösungsansatz: Algorithmus von Prim

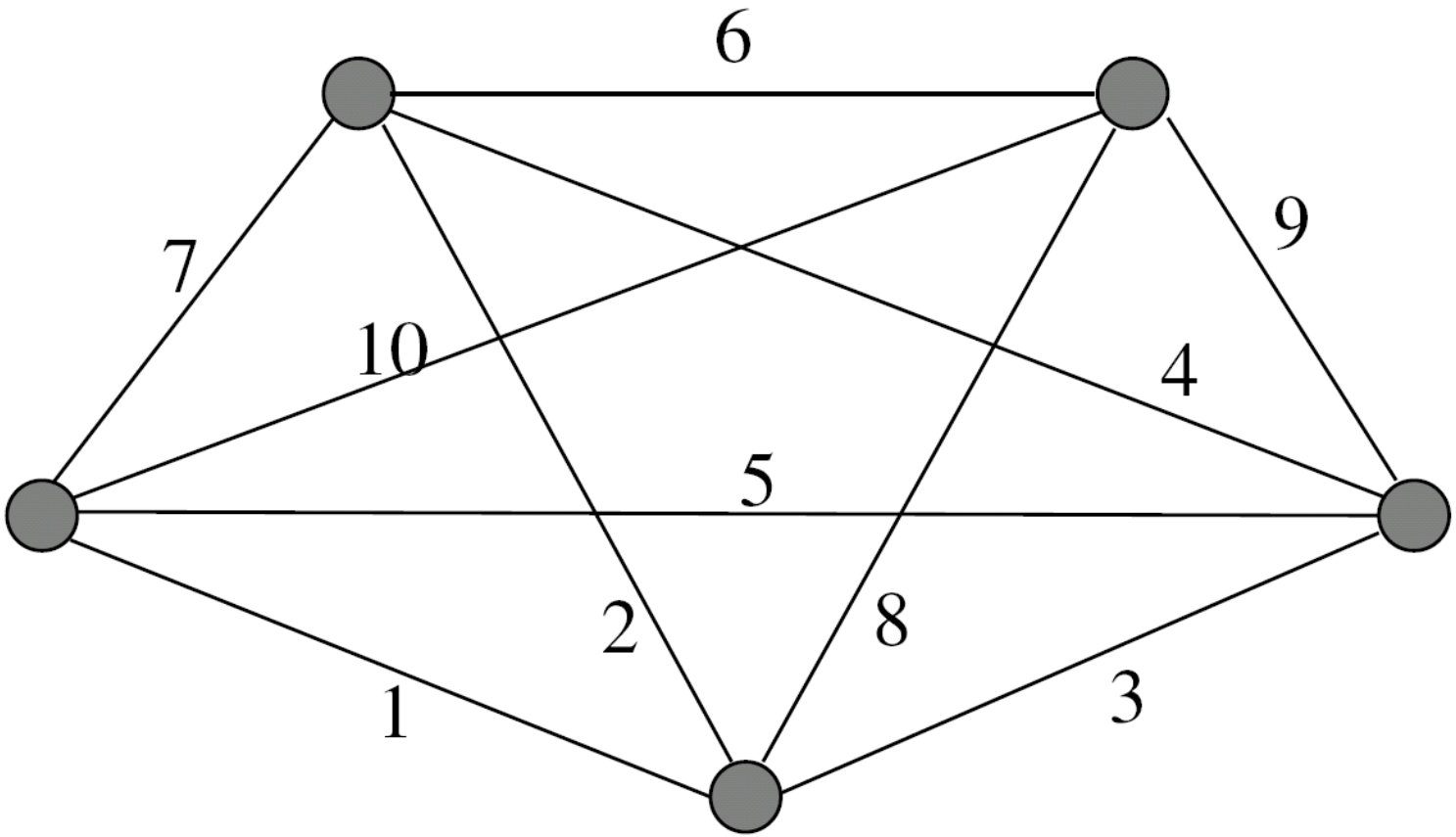
- Vorgehensweise
  - Schrittweise Verfeinerung
  - Aufbau eines aufspannenden Baumes durch Hinzufügen von Kanten
- Greedy-Muster
  - Jeweils kostengünstigste geeignete Kante als Erweiterung

Aufspannender minimaler Baum

Pseudocode:

```
// Teilbaum B besteht anfangs aus einem
// beliebigen Knoten
while [ B noch nicht G_V aufspannt ]
do [ suche kostengünstige von B ausgehende Kante ];
    [ füge diese Kante zu B hinzu ];
od
```

Beispiel



Analyse

Theorem (Terminierung)

Der Algorithmus von Prim terminiert nach endlicher Zeit.

Beweis:

Einfache Schleifenanalyse. □

Theorem (Laufzeit)

Wird für die Implementierung ein Fibonacci-Heap benutzt, so hat der Algorithmus von Prim eine Laufzeit von $O(|E| + |V| \log |V|)$.

Ohne Beweis

Analyse

Theorem (Korrektheit)

Ist G ein verbundener ungerichteter gewichteter Graph, so berechnet der Algorithmus von Prim einen minimalen Spannbaum von G.

Beweis:

Betrachte

```
while [ B noch nicht G_V aufspannt ]
do [ suche kostengünstige von B ausgehende Kante ];
     [ füge diese Kante zu B hinzu ];
od
```

Beobachtung:

- B ist am Ende ein Baum
- B ist am Ende ein Spannbaum

Analyse

Noch zu zeigen: B ist am Ende ein minimaler Spannbaum.

```
while [ B noch nicht G_V aufspannt ]
do [ suche kostengünstige von B ausgehende Kante ];
     [ füge diese Kante zu B hinzu ];
od
```

Sei B‘ ein minimaler Spannbaum von G und B≠B‘.

Betrachte den Zeitpunkt in der Hauptschleife, an dem sich die Konstruktion von B von B‘ unterscheidet.

Sei e die Kante, die dann zu B hinzugefügt wird.

Sei V_1 die Menge der Knoten, die schon in B sind und $V_2=V\backslash V_1$

Analyse

Da B‘ ein minimaler Spannbaum ist, gibt es eine Kante e‘, die V_1 mit V_2 verbindet.

Da im Algorithmus stets eine günstigste Kante gewählt wird, muss gelten $g(e) \leq g(e')$.

Tauschen wir in B‘ die Kante e‘ durch e erhalten wir also einen minimalen Spannbaum, der nicht mehr kostet als B‘, es folgt $g(e)=g(e')$.

Induktiv folgt damit die Korrektheit. □

Algorithmen und Datenstrukturen

11.2 SPANNBÄUME
ZUSAMMENFASSUNG

Zusammenfassung

- Spannbaum: Subgraph, der alle Knoten und genau |V|-1 Kanten enthält
- Minimaler Spannbaum: Spannbaum mit minimalem Gesamtgewicht
- Algorithmus von Prim
  - *Idee:* Wähle stets günstigste vom aktuellen Baum ausgehende Kante aus