Teil III: Kreise, Rundreisen, Transportnetze

1. Kreise
2. Traveling Salesman Problem
3. Transportnetzwerke


Kreise, Rundreisen und Transportnetzwerke

- Euler-Kreis
- Hamilton-Kreis
- Traveling Salesman Problem
- Flüsse in Transportnetzwerken
- Algorithmus von Ford / Fulkerson
- Schnitte in Graphen
- Min-Cut-Max-Flow - Theorem
- Matching
- Netzplantechnik, Kritische-Pfad-Analyse
- Kritische Aktivitäten
- Petri-Netze



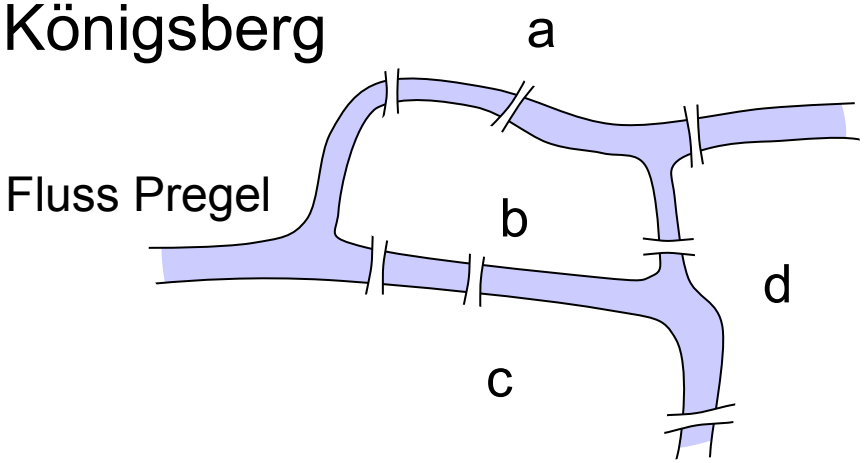




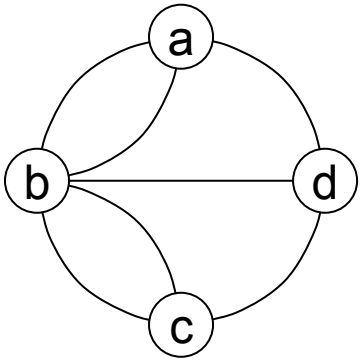
Euler-Kreis

- Beispiel: Brückenproblem

Königsberg



schematisiert:



Gibt es einen Weg, um auf einem Spaziergang durch Königsberg alle Brücken genau einmal zu überqueren?

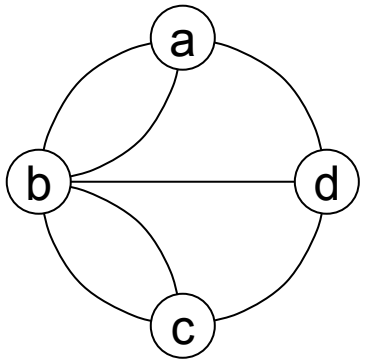
Euler-Kreis (1)

- Satz (Euler)
  - Ein Graph G (zusammenhängend, Mehrfachkanten diesmal erlaubt) besitzt einen Euler-Kreis genau dann, wenn alle Knotengrade gerade sind.

- Zurück zur Fragestellung:

  Gibt es einen Weg, um auf einem Spaziergang durch Königsberg alle Brücken genau einmal zu überqueren?

schematisiert:



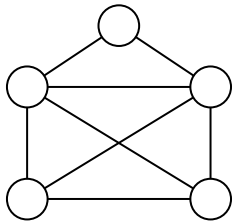
Die Grade jedes Knoten in diesem Beispiel sind ungerade.

Daraus folgt:
Es existiert kein Euler-Kreis.

Für dieses Problem existiert ein effizienter Algorithmus.

Euler-Tour (2)

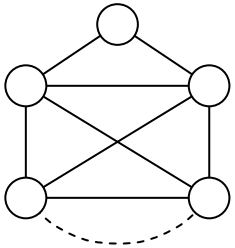
- wie Euler-Kreis, jedoch Anfangs- und Endknoten müssen nicht übereinstimmen.



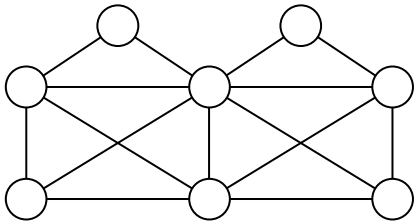
→ ungerader Graph

Es gibt eine Euler-Tour.

- Durch Hinzufügen einer Kante bekommen alle Kanten einen geraden Grad. Also gibt es im ergänzten Graphen einen Eulerkreis.



Euler-Tour (3)



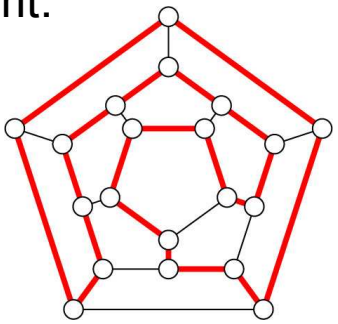
- Gibt es für das "Doppelhaus des Nikolaus" eine Euler-Tour, d.h. kann man dies durchgehend und ohne Kantenwiederholung zeichnen?

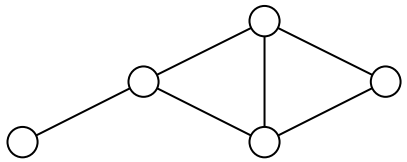
- Nein, denn es gibt vier Knoten mit ungeradem Grad.

Hamiltonkreis

- Effiziente Lösbarkeit und ineffiziente Lösbarkeit liegen „dicht“ beieinander.
- Beispiel
  - Man kann effizient entscheiden, ob ein Graph einen Eulerkreis enthält.
    - Teste alle Knoten auf geraden Grad:
    - Für jeden von n Knoten werden alle n – 1 anderen Knoten auf Nachbarschaft getestet.
    - Testalgorithmus hat Komplexität $n \cdot (n - 1) \leq n^2 - n$, also Polynom, also effizient.
  - Man kann aber nicht effizient entscheiden, ob ein Graph einen Hamiltonkreis enthält.
    - Ein **Hamiltonkreis** ist ein Kreis, bei dem jeder Knoten genau einmal vorkommt.

Graph mit Hamiltonkreis





Graph ohne Hamiltonkreis

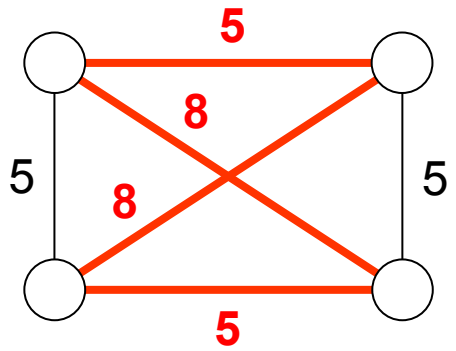
Wege in Graphen

- Anstatt Wege mit nur wenigen bzw. kostengünstigen Kanten zu nutzen, kann man nach Wegen suchen, die **alle Kanten** bzw. **alle Knoten** beinhalten.

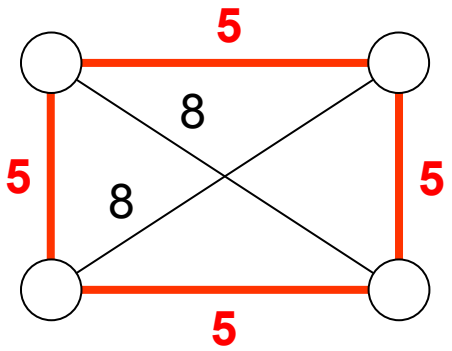
- Ein **Eulerscher Weg** oder eine **Euler Tour** in einem Graphen G = (V, E) ist ein Weg, der jede Kante von E genau einmal enthält.
- Ein **Eulerscher Kreis** in G = (V, E) ist ein Kreis, der jede Kante von E genau einmal enthält.
- Ein **Hamiltonscher Weg** in einem Graphen G = (V, E) ist ein Weg, der jeden Knoten von V genau einmal enthält.
- Ein **Hamiltonscher Kreis** in G = (V, E) ist ein Kreis, der jeden Knoten von V genau einmal enthält (ausser Anfangspunkt).

Traveling Salesman Problem (TSP, Rundreiseproblem)

- Finde unter allen Hamiltonkreisen einen mit minimaler Summe der Kantenbewertungen („Finde eine (kosten-) minimale Rundreise“)



Rundreise mit Kosten 26



Rundreise mit Kosten 20

- Das Problem ist nicht effizient lösbar; auch dann nicht, wenn das Vorliegen von Hamiltonkreisen gesichert ist.

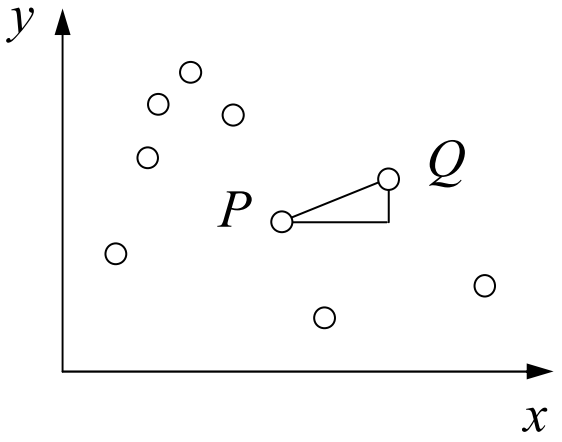
Traveling Salesman Problem (TSP, Rundreiseproblem) (1)

Ansatz:

- Approximierte Lösung von TSP durch effiziente Heuristik.
- Heuristik: Auffinden einer „guten Lösung“ mit „begrenzten“ Möglichkeiten.

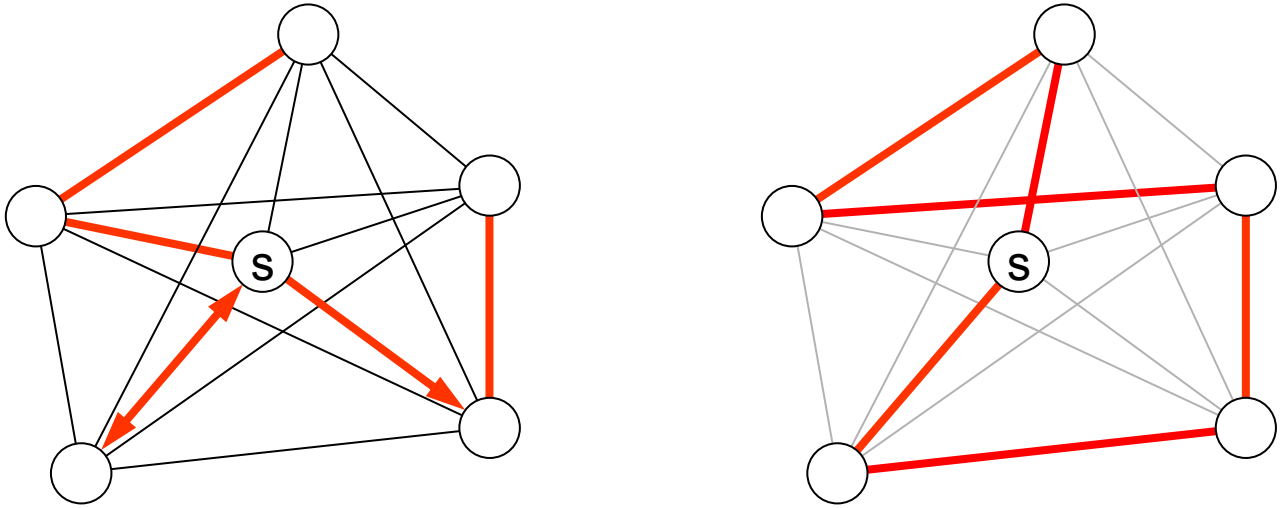
Euklidisches TSP:

- Knoten des Graphen liegen in Ebene, die Entfernungen sind Euklidische Abstände, Graph ist vollständig.



$$P(x_1 \mid y_1)$$
$$Q(x_2 \mid y_2)$$
$$d(P,Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

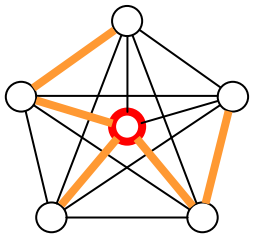
Traveling Salesman Problem (TSP, Rundreiseproblem) (2)

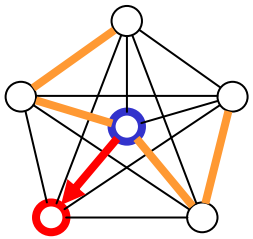


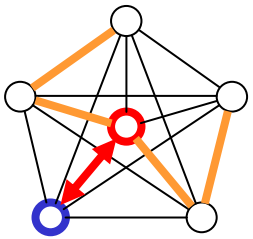
Heuristik für Euklidisches TSP:

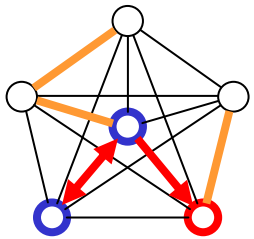
- Aufbau eines minimal spannenden Baums mit Kruskal Algorithmus.
- Abfahren bis jeweils ein Knoten von Grad 1 erreicht ist,
  dann zum zweiten mal abfahren bis ein noch unbesuchter Knoten erreicht ist, „kurzschließen“ und Abfahren fortsetzen.

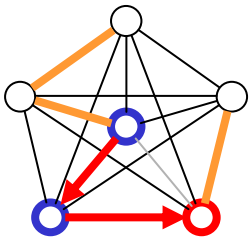
Traveling Salesman Problem (TSP, Rundreiseproblem) (2)

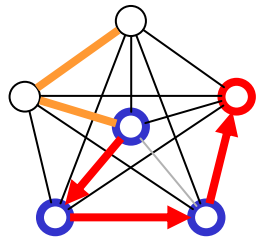

Start

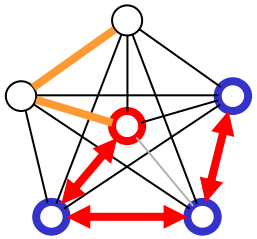

Abfahren
bis Knoten von Grad 1

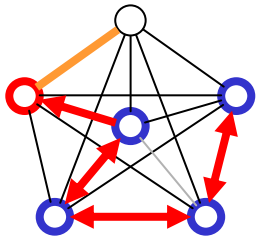

zurück abfahren

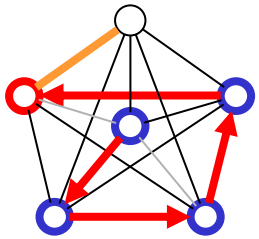

noch unbesetzter
Knoten

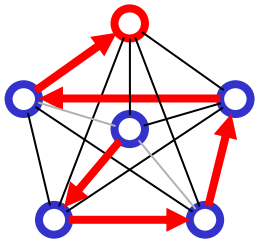

"kurzschließen"

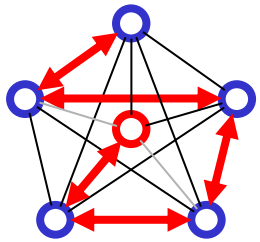

Abfahren fortsetzen
bis Knoten von Grad 1

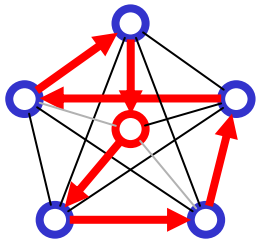

zurück abfahren


noch unbesetzter
Knoten


"kurzschließen"


Abfahren fortsetzen
bis Knoten von Grad 1


zurück abfahren


"kurzschließen"

Heuristik für Euklidisches TSP:

- Aufbau eines minimal spannenden Baums mit Kruskal Algorithmus.
- Abfahren bis jeweils ein Knoten von Grad 1 erreicht ist,
  dann zum zweiten mal abfahren bis ein noch unbesuchter Knoten erreicht ist, „kurzschließen“ und Abfahren fortsetzen.

Transportnetzwerke

- Flüsse in Transportnetzwerken
- Algorithmus von Ford / Fulkerson
- Schnitte in Graphen
- Min-Cut-Max-Flow - Theorem
- Matching
- Netzplantechnik, Kritische-Pfad-Analyse
- Kritische Aktivitäten
- Petri-Netze








Flüsse in Transportnetzwerken

- Gegeben sei ein gerichteter Graph G = (V,E) mit Kantenbewertungen cap: $E \rightarrow \mathbb{R}_+$
- Anwendung: z.B. Rohrleitungsnetz, elektrisches Leitungsnetz, Transportwege
- cap-Wert sind Kapazitäten (z.B. maximale Werte in Liter/sec, PKW/h, bit/sec, Ampere

➢ Zwei Knoten besonders hervorgehoben

s: Quelle t: Senke

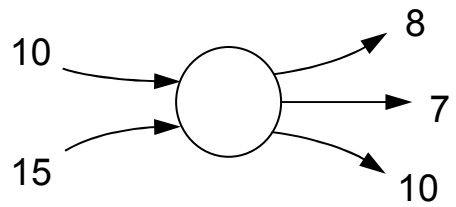
Flüsse in Transportnetzwerken (1)

- Problem:

  Bestimmung eines maximalen Flusses von s nach t

- Fluss $f: E \to \mathbb{R}_+$ wobei $f(e) \leq cap(e) \quad \forall e \in E$

  und Summe ankommender Flüsse = Summe hinausgehender Flüsse (Konservierungsbedingung) für alle Knoten außer s und t.

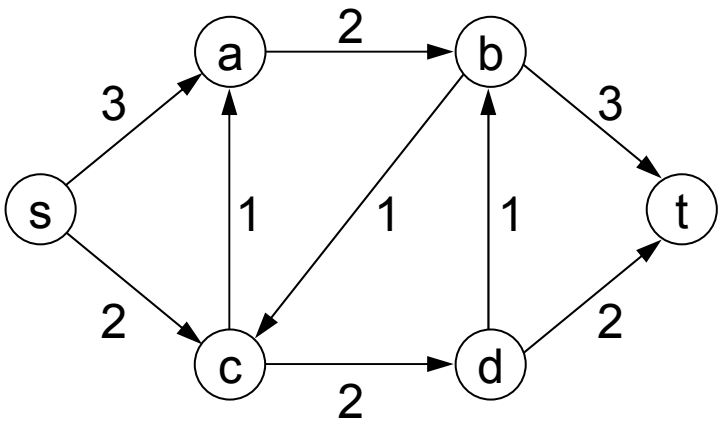


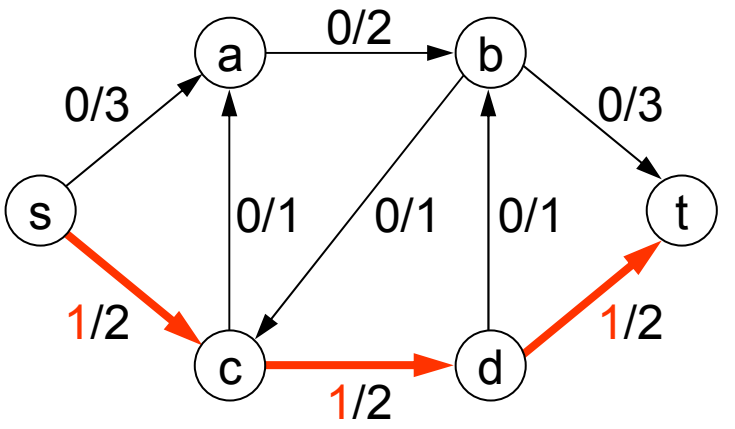
- max! $\sum_{(s,y) \in E} f(s,y)$

d.h. maximiere die aus s abfließende Menge

Flüsse in Transportnetzwerken (3)

- Beispiel:





Kantenwerte sind Kapazitäten

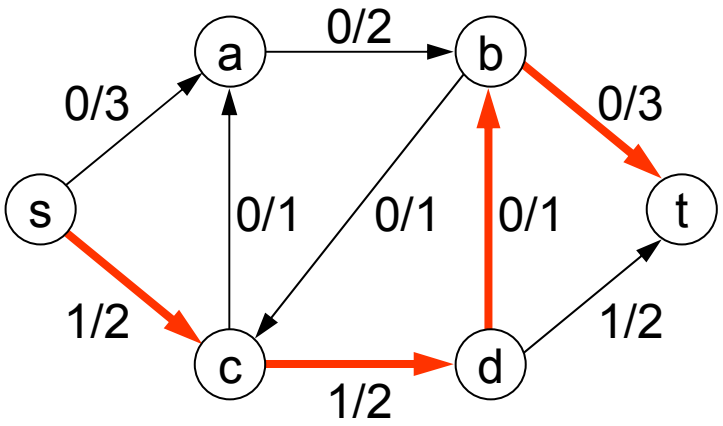
Fluss mit Wert 1 = Stärke 1

Summe aller Zuflüsse = Summe aller Abflüsse

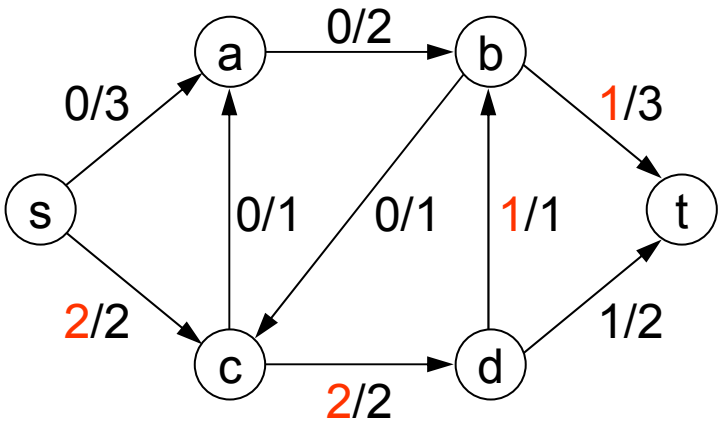
gilt in allen Knoten ausser s und t

Notation

Flüsse in Transportnetzwerken (4)

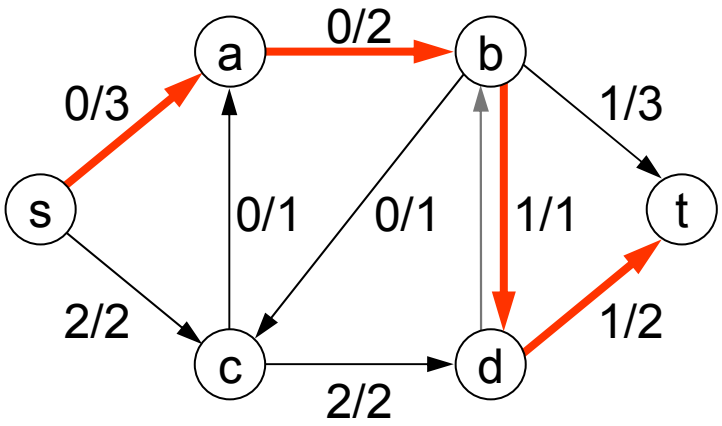


Verbesserung durch
flusserhöhende Pfade

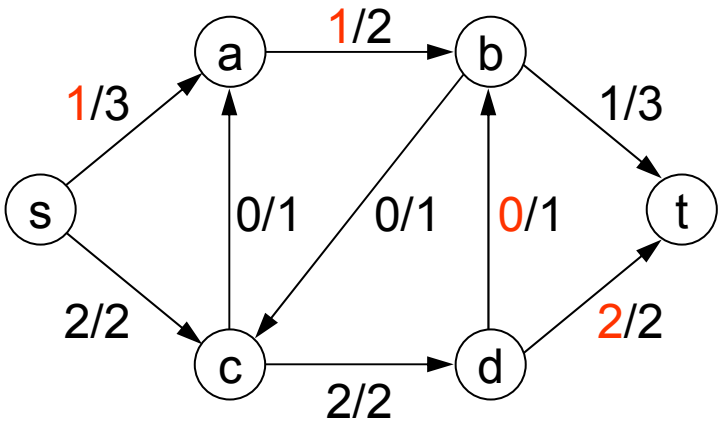


neuer Fluss mit Stärke 2

Flüsse in Transportnetzwerken (5)



Ein positiver Fluss kann ganz oder teilweise entgegen der Kantenrichtung „zurückgedrückt“ werden!



Es resultiert ein Fluss der Stärke 3

Flüsse in Transportnetzwerken (6)

- Ein flusserhöhender Pfad bzgl. eines Flusses f ist ein Weg

  $s = i_0, i_1, \ldots, i_k = t$ (Knotenfolge!) so, dass

  - $f(i_j, i_{j+1}) < cap(i_j, i_{j+1})$ falls $(i_j, i_{j+1}) \in E$

    (auf „Vorwärtskanten“ ist die Kapazität nicht ausgeschöpft)

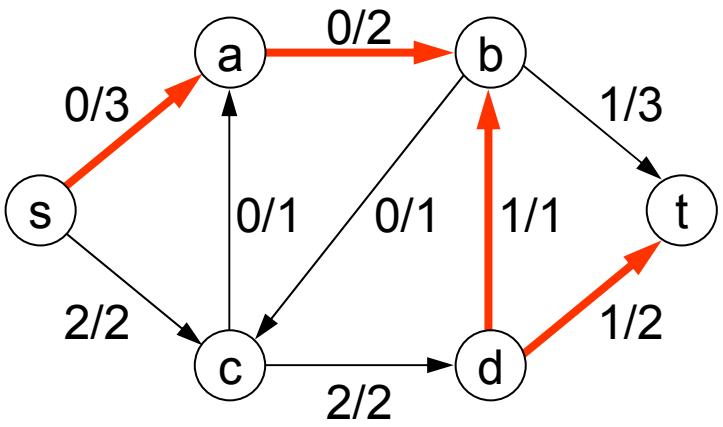
  - $f(i_j, i_{j+1}) > 0$ falls $(i_{j+1}, i_j) \in E$

    (auf „Rückwärtskanten“, gibt es einen Fluss, der „zurückgeschoben“ werden kann)

- Restkapazität m = Minimum über die Werte $cap(i_j, i_{j+1}) - f(i_j, i_{j+1})$ (für Vorwärtskanten)

  und die Werte $f(i_j, i_{j+1})$ (für Rückwärtskanten)

Flüsse in Transportnetzwerken (7)



s, a, b, d, t

(b,d) $\notin$ E, aber

(s,a) $\in$ E mit 0 < 3
(a,b) $\in$ E mit 0 < 2
(d,b) $\in$ E mit 1 > 0
(d,t) $\in$ E mit 1 < 2

Also ist s, a, b, d, t ein flusserhöhender Pfad. Eine Erhöhung um Restkapazität = min {3 – 0, 2 – 0, 1, 2 – 1} = 1 ist möglich

Flüsse in Transportnetzwerken (8)

- Auf einem flusserhöhenden Pfad wird der Zusatzfluss f′ angesetzt als

  - $f'(i_j, i_{j+1}) = m$ für $(i_j, i_{j+1}) \in E$ Vorwärtskante
  - $f'(i_j, i_{j+1}) = -m$ für $(i_{j+1}, i_j) \in E$ Rückwärtskante

Algorithmus von Ford / Fulkerson

- **Satz (Ford / Fulkerson)**

  Ein Fluss ist genau dann maximal, wenn es keinen flusserhöhenden Pfad gibt.

- Daraus resultiert der Algorithmus von Ford und Fulkerson zur Flussmaximierung

Algorithmus von Ford / Fulkerson (1)

➢ Algorithmus zur Flussmaximierung

1) Beginne mit 0-Fluss f, d. h. setze f(e) := 0 $\quad \forall e \in E$
2) Finde einen beliebigen flusserhöhenden Pfad
3) Finde einen Zusatzfluss f′ entlang dieses Pfades mit Stärke = Restkapazität
4) Addiere den Zusatzfluss zum aktuellen Fluss, d. h. f := f + f′
5) Wiederhole dies, bis es keinen flusserhöhenden Pfad mehr gibt

Algorithmus von Ford / Fulkerson (2)

➢ Algorithmus zur Flussmaximierung

1. Startsetzungen („Initialisierungen“)

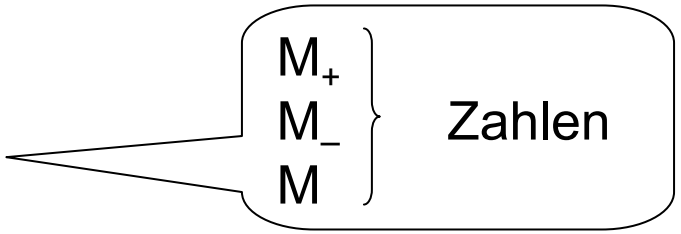
   $f(e) := 0, \forall\ e \in E$

2. Wiederhole solange $\exists$ flusserhöhenden Pfad E'

   $M_+ := \min_{\substack{e \in E' \\ e \text{ Vorwärtskante}}} (cap(e) - f(e))$

   $M_- := \min_{\substack{e \in E' \\ e \text{ Rückwärtskante}}} f(e)$

   $M := \min(M_+, M_-)$

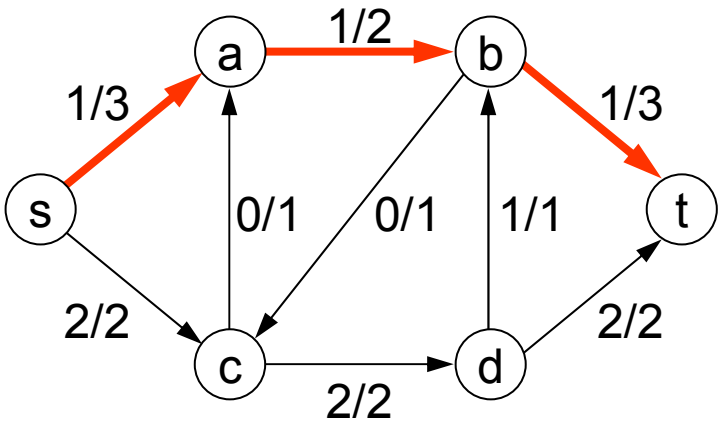
   Wiederhole $\forall\ e \in E'$:

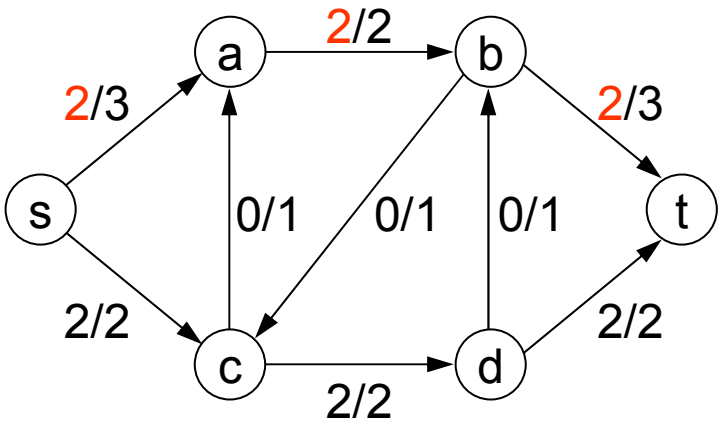
   Wenn e Vorwärtskante dann $f(e) := f(e) + M$

   Wenn e Rückwärtskante dann $f(e) := f(e) - M$

3. Ausgabe $f(e), \forall\ e \in E$

Algorithmus von Ford / Fulkerson (3)





Es gibt keinen flusserhöhenden Pfad!

(s, c) liegt nicht auf flusserhöhendem Pfad, da

$f(s, c) = 2 \not< 2 = cap(s, c)$
s, a, b,… für (a, b) ist $f(a, b) = 2 \not< 2 = cap(a, b)$
s, a, c,… für (a, c) ist $f(c, a) \not> 0$

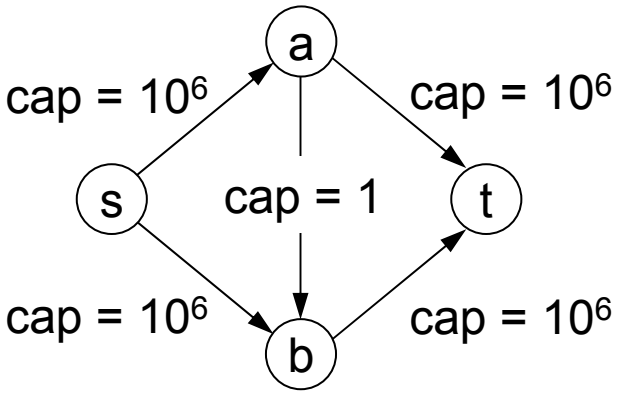
Algorithmus von Ford / Fulkerson (4)

- Animation siehe http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/maxflow/Maxflow.shtml

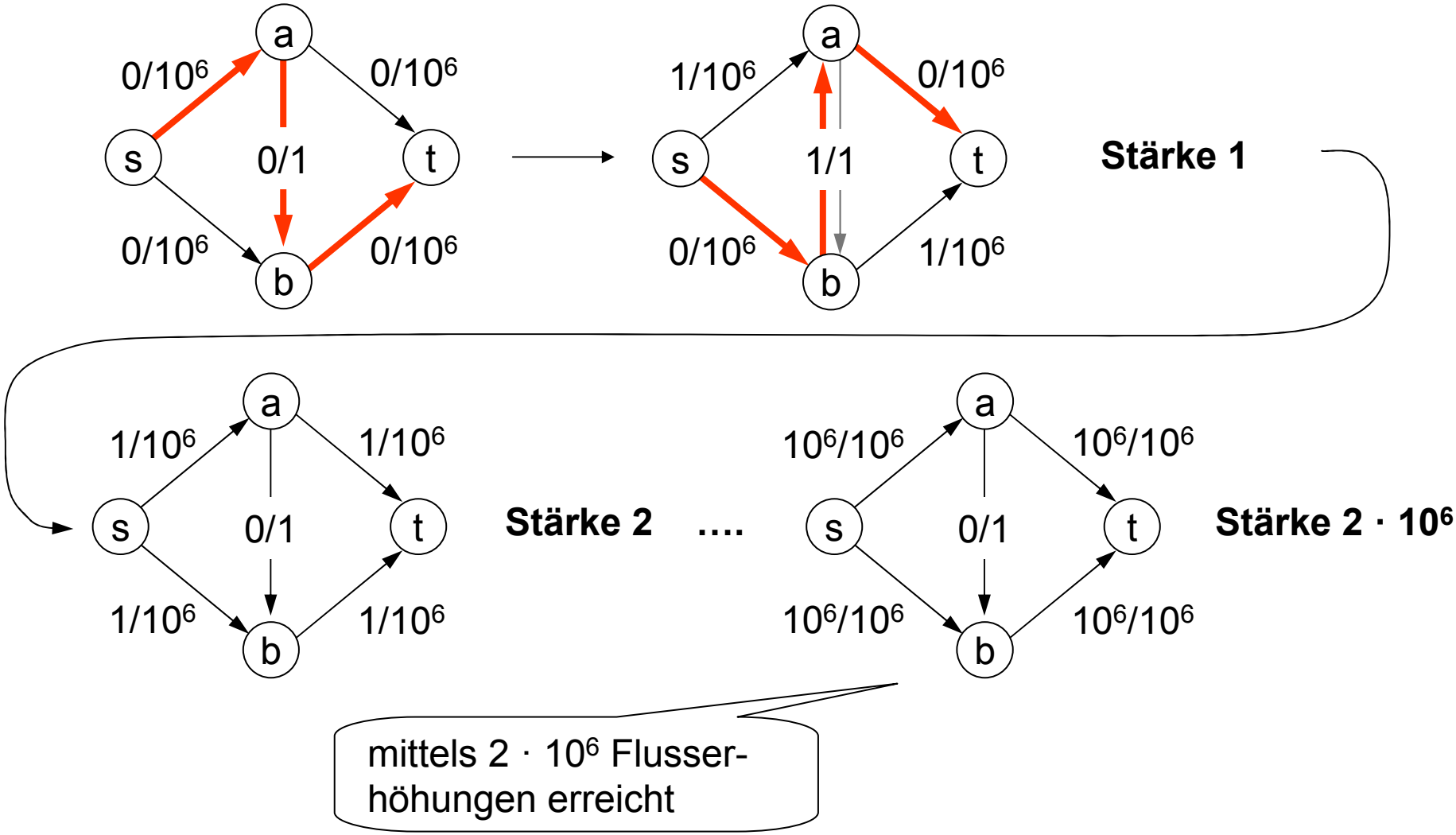
Algorithmus von Ford / Fulkerson (5)

- Es gibt günstige und ungünstige Wahlen für flusserhöhende Pfade

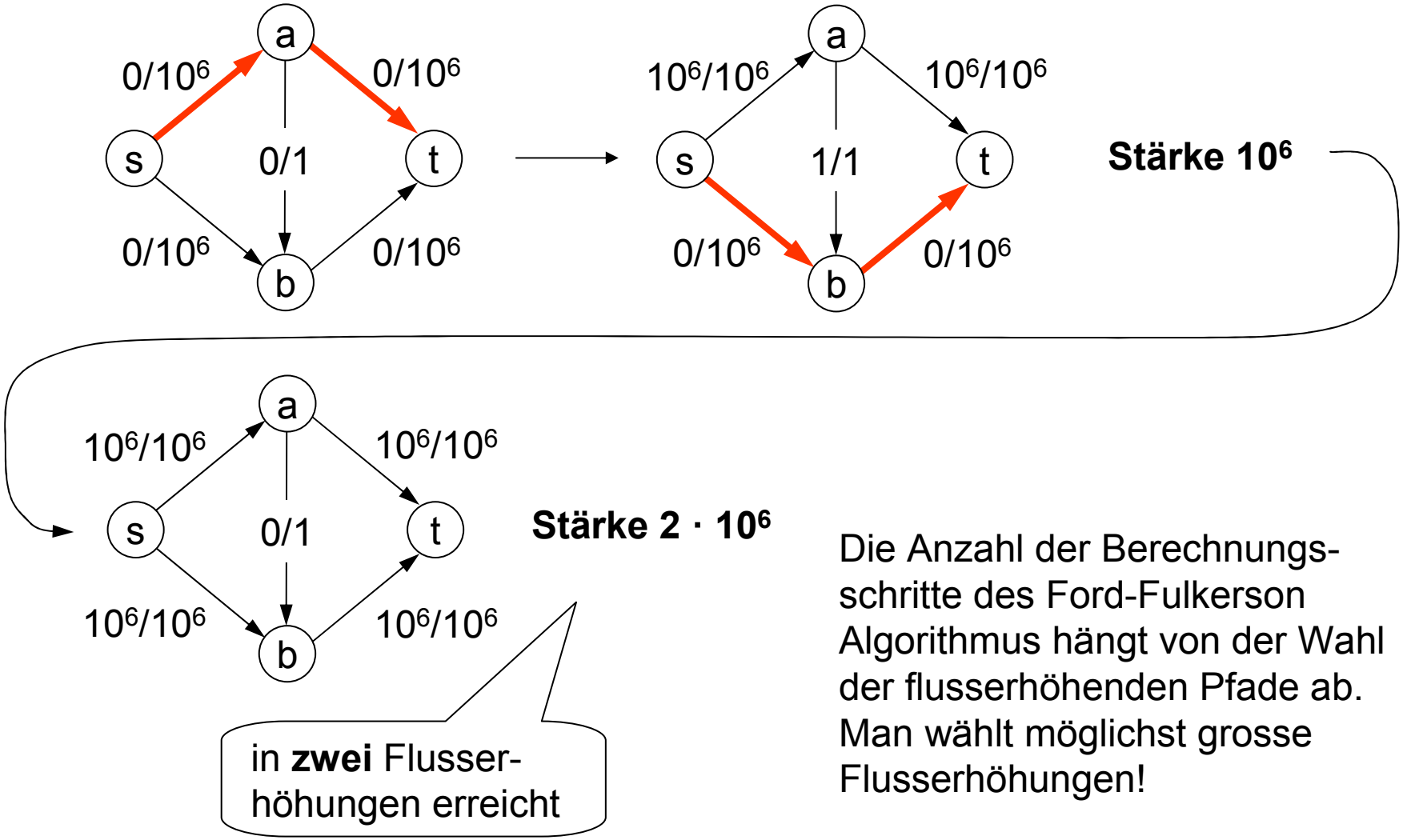
- Beispiel



Algorithmus von Ford / Fulkerson (6)



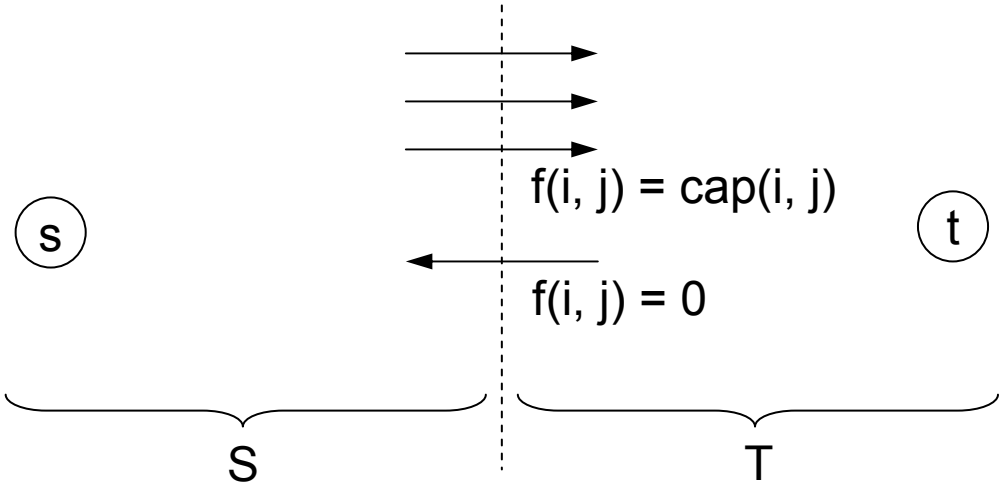
Algorithmus von Ford / Fulkerson (7)



Die Anzahl der Berechnungs-schritte des Ford-Fulkerson Algorithmus hängt von der Wahl der flusserhöhenden Pfade ab. Man wählt möglichst grosse Flusserhöhungen!

Schnitte in Graphen

- Kann man für den maximalen Flusswert eine obere Schranke angeben, ohne Flussmaximierung durchzuführen?



- Idee: der maximale Fluss kann nicht grösser sein als die Kantenkapazitäten entlang einer Trennlinie durch die Kanten („Schnitt“)

Schnitte in Graphen (1)

- Schnitt:

  Aufteilung der Knotenmenge V in zwei disjunkten Mengen S, T, d. h.

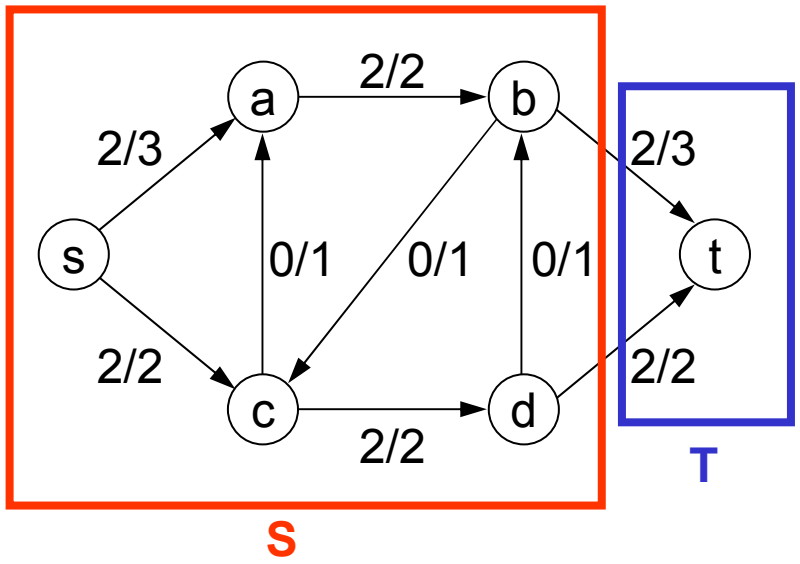
$$S \cap T = \emptyset, \text{ so dass } s \in S \text{ und } t \in T$$

  Schnittkapazität oder Wert eines Schnitts:

$$cap(S, T) = \sum_{\substack{(i, j) \in E \\ \text{mit } i \in S,\ j \in T}} cap(i, j)$$

Schnitte in Graphen (2)

➢ Beispiel:



$$cap(S, T) = cap(b, t) + cap(d, t) = 3 + 2 = 5$$

	Wert eines Flusses	≤	Wert eines Schnitts
⇒	Wert eines Flusses	≤	minimaler Wert eines Schnitts
⇒	Maximaler Wert eines Flusses	≤	minimaler Wert eines Schnitts

Min-Cut-Max-Flow - Theorem

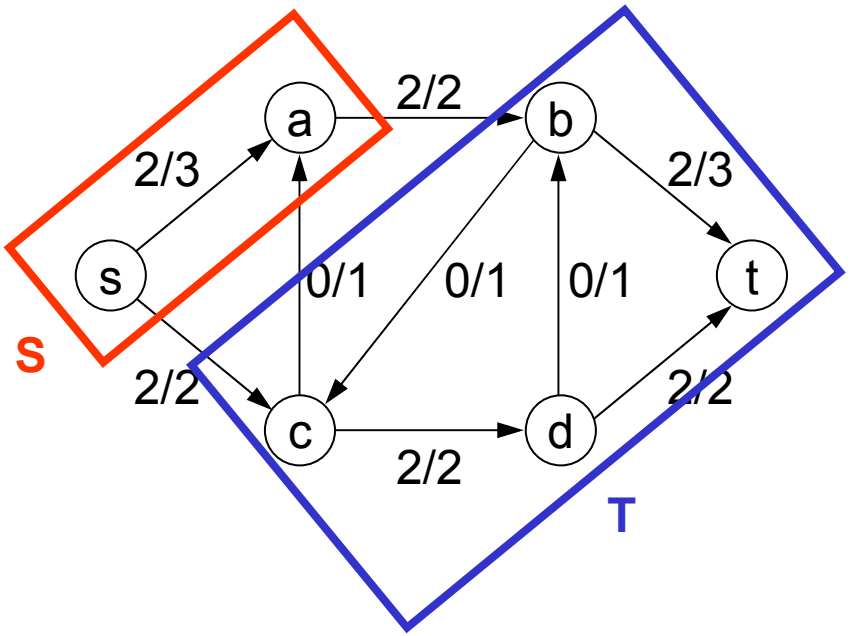
- Der Wert eines maximalen Flusses ist gleich (= !) dem Wert eines minimalen Schnitts

- Bestimmung eines minimalen Schnitts

  - Bestimmung eines maximalen Flusses mit Ford / Fulkerson Algorithmus

  - Bestimmung aller von s aus erreichbaren Knoten über Kanten mit Restkapazität > 0. Die so erreichbaren Knoten gehören zu S, alle anderen zu T.

Min-Cut-Max-Flow – Theorem (1)



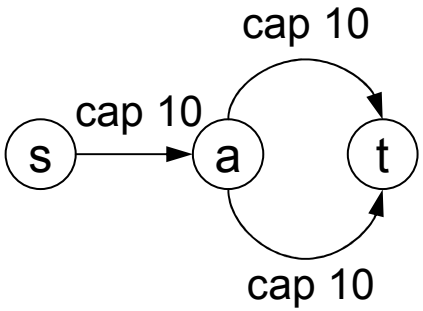
$$cap(S, T) = cap(a, b) + cap(s, c) = 2 + 2 = 4$$

maximaler Flusswert

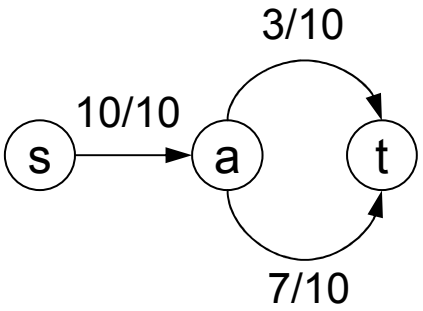
Min-Cut-Max-Flow – Theorem (2)

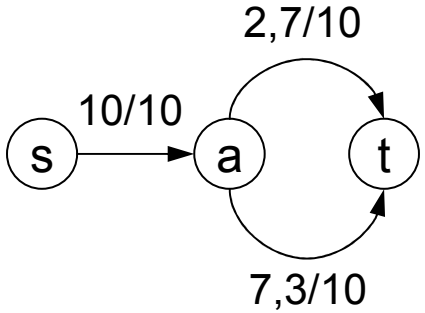
- Wenn alle Kantenkapazitäten ganzzahlig sind, ist unter den maximalen Flüssen auch ein ganzzahliger.
  D.h. die Flusswerte auf **allen** Kanten sind ganzzahlig.

- Aber: es kann auch maximale Flüsse mit gebrochenen Werten geben



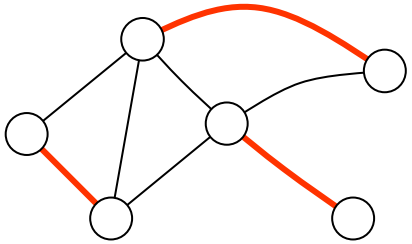
Maximale Flüsse der Stärke 10





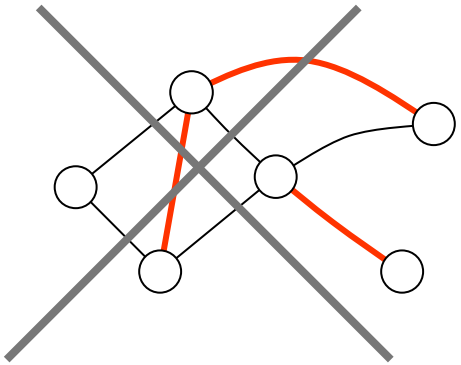
Matching

- Gegeben sei ein Graph G = (V, E).
- Ein Matching ist eine Teilmenge von E, $M \subseteq E$,
  so dass keine zwei Kanten in M denselben Knoten gemeinsam haben.



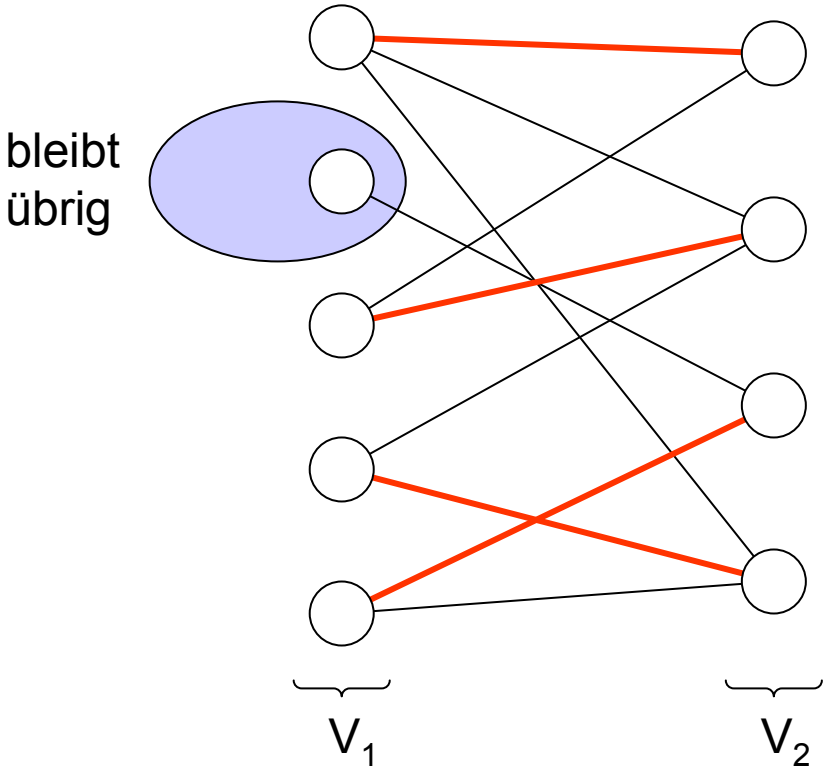
Matching M

Dies ist ein **perfektes** Matching,
d.h. **jeder** Knoten kommt in M vor



kein Matching

Matching bei bipartiten Graphen

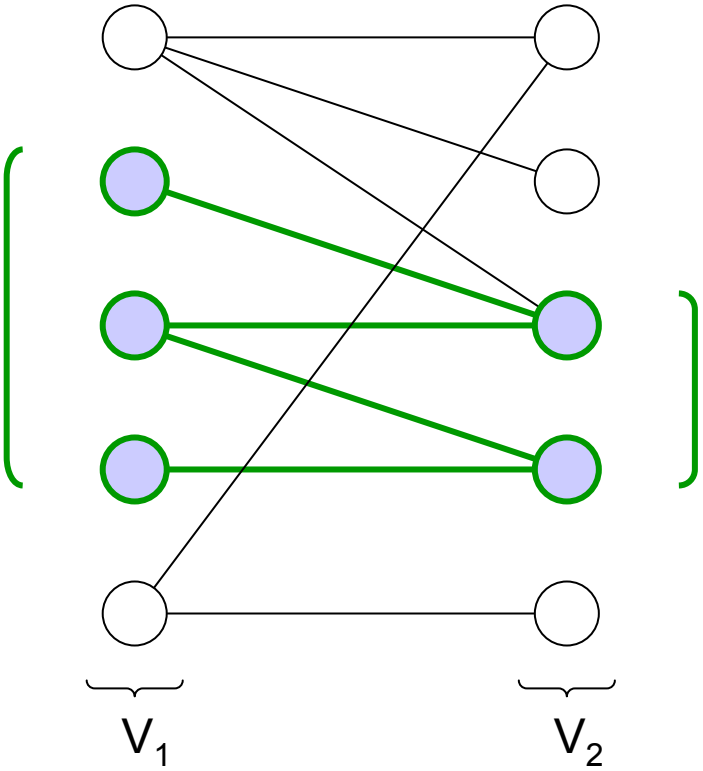


Matching

Ein perfektes Matching ist hier nicht möglich, da $|V_1| > |V_2|$ ist (d. h. Knotenmenge links > Knotenmenge rechts).

$V = V_1 \cup V_2$

"Heiratssatz"



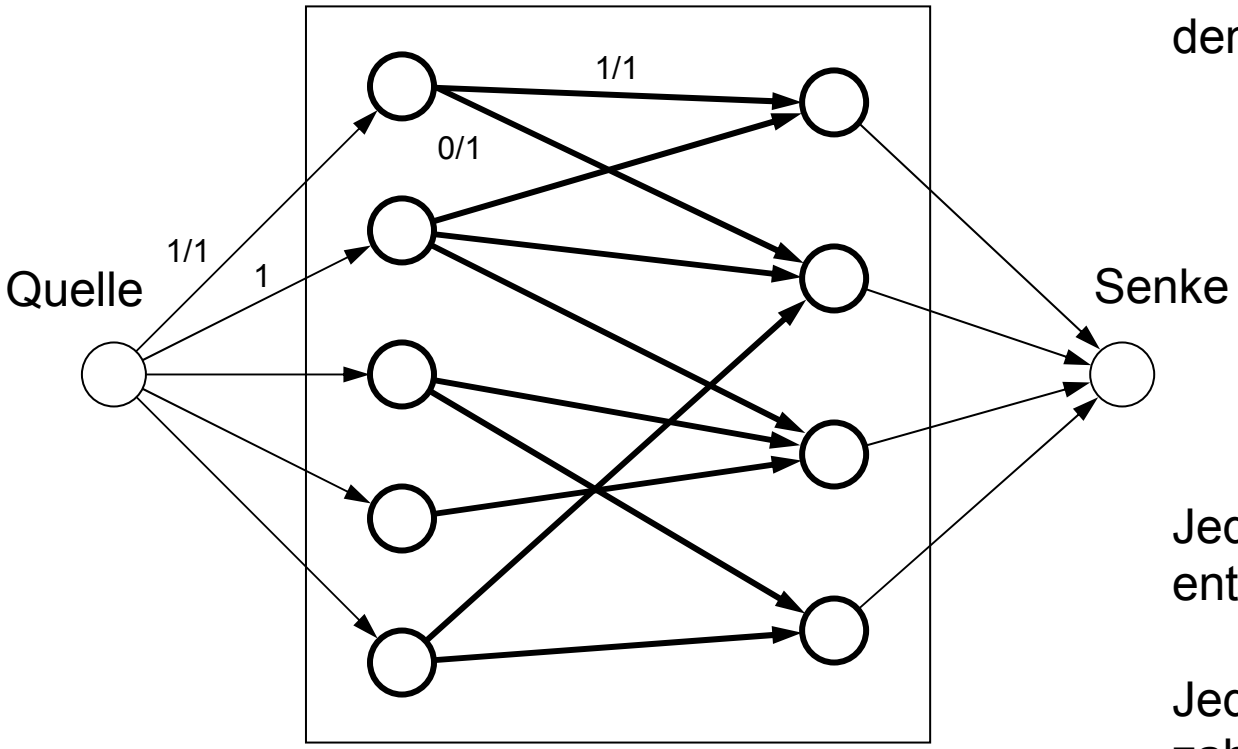
Bei bipartiten Graphen mit $|V_1| = |V_2|$ ist ein perfektes Matching genau dann möglich, falls für alle $A \subseteq V_1$ gilt: $|N(A)| \geq |A|$.

(Menge der Nachbarn von A ≥ A selbst.)

Die drei Knoten auf der linken Seite haben keine Verbindung zu anderen als den zwei Knoten auf der rechten Seite. Daher ist kein perfektes Matching möglich.

Maximales Matching (1)

- Gegeben sei ein bipartiter Graph.
- Bestimme ein maximales Matching.



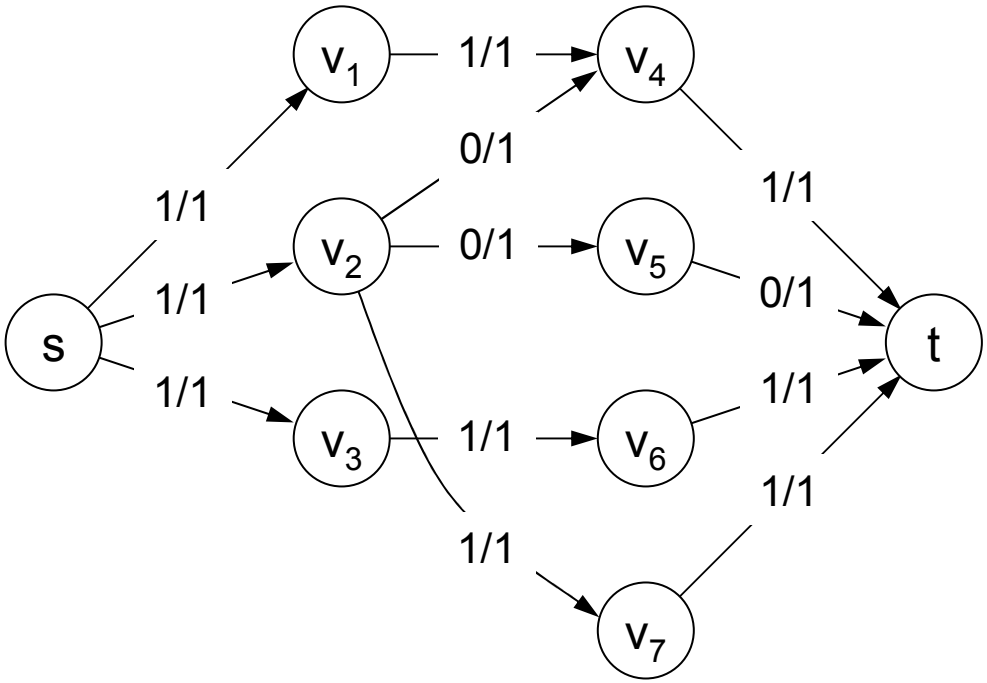
Bipartiter Graph

Füge links eine Quelle und rechts eine Senke hinzu. Setze alle cap-Werte auf 1. Ford / Fulkerson berechnet den maximalen Durchfluss.

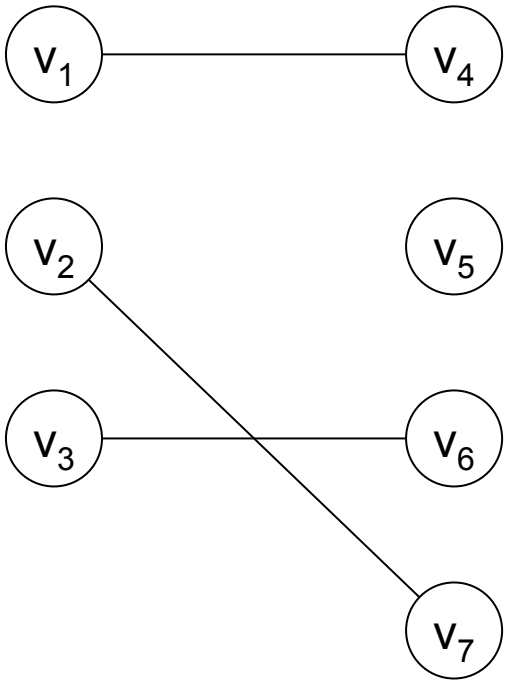
Jeder ganzzahlige Fluss entspricht einem Matching

Jeder maximale ganz-zahlige Fluss entspricht einem maximalen Matching

Maximales Matching (2)



Maximaler ganzzahliger Fluss
der Stärke 3

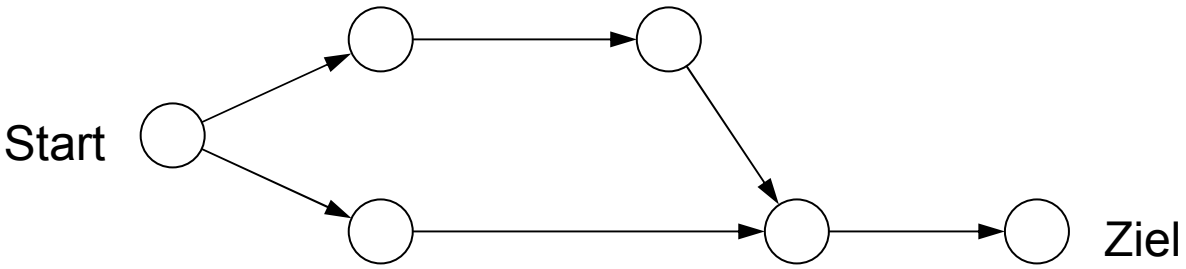


Matching mit 3 Kanten

Netzplantechnik, Kritische-Pfad-Analyse

- (Critical Path Method – CPM,
  Program Evaluation and Review Technique – PERT)

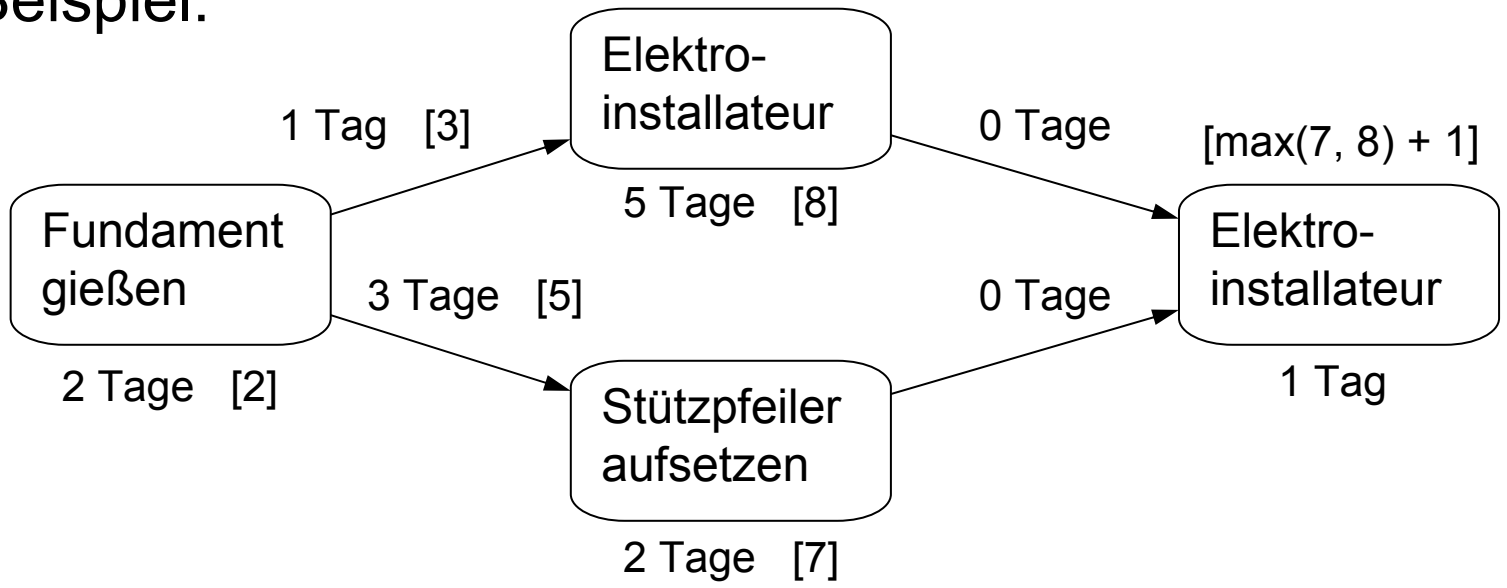
- Gegeben sei ein gerichteter, azyklischer Graph (DAG) mit gegebenen Start- und Zielknoten, wobei
  - Knoten ≙ Aktivitäten / Tätigkeiten!
  - Kanten ≙ Zeitliche / Kausale Abfolgen!



Netzplantechnik, Kritische-Pfad-Analyse (1)

- Den Knoten und Kanten sind Werte $\in \mathbb{R}_+$ zugeordnet (Zeiten für die betreffenden Aktivitäten bzw. Übergangszeiten).

- Beispiel:



- Frage: Wieviel Zeit vergeht vom Start bis zum Ziel?

„Durchrechnen“ eines Netzplans

- Addiere die Werte entlang eines Pfades, beginnend beim Start-Knoten.
- Bei Knoten mit mehreren Vorgängerknoten, übernehme den maximalen Wert.
- Wert des Zielknotens ≙ Zeitdauer für das gesamte Projekt.

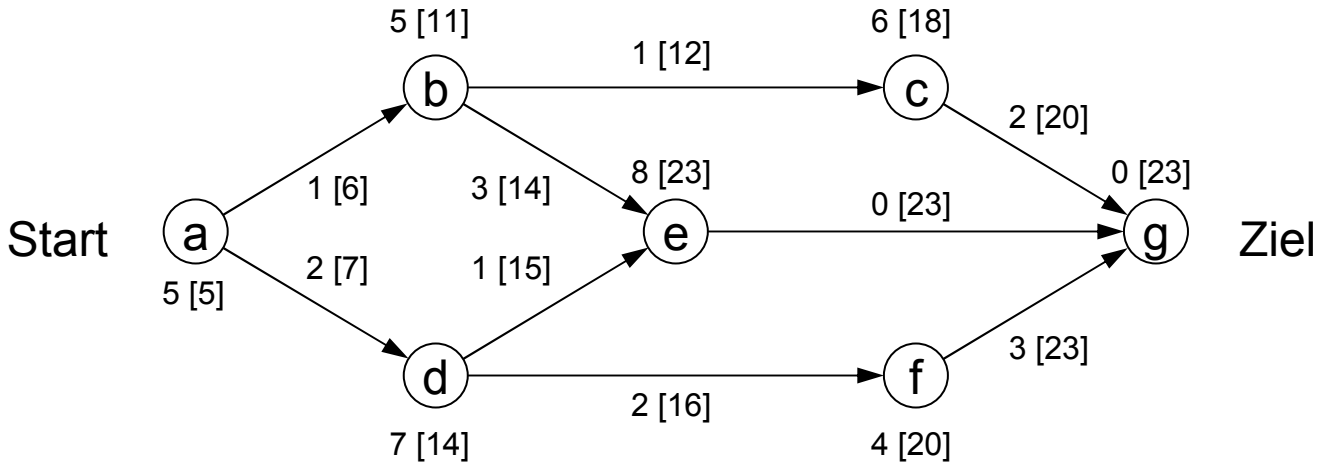
- Rückwärtsanalyse, vom Zielknoten ausgehend, verfolge welche Pfade bei der Maximumsbildung relevant waren

→ Kritischer Pfad oder kritische Pfade.

Kritische Aktivitäten

- Aktivitäten, die auf kritischen Pfaden liegen, heißen **kritische Aktivitäten**.

- Beispiel:



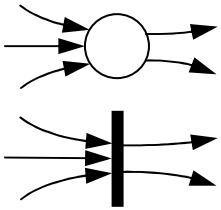
- Kritische Aktivitäten sind: a, d, e, f, (g)

Hier Dummy-Knoten
(strukturell verwendet, d. h. Zeit 0)

Petri-Netze

- Petri-Netze sind Spezielle Graphen zur Berechnung dynamischer Abläufe (die eine feste Grundstruktur besitzen).
- Anwendungen: Workflow, Betriebssystem

- Zwei Sorten von Knoten wechseln sich ab (bipartiter Graph):



sog. Stellen

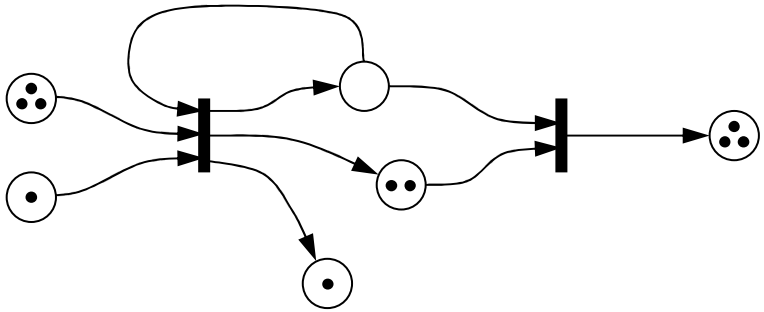
sog. Transitionen

- G = (V,E), gerichteter Graph, wobei $V = S \cup T$

Stellen

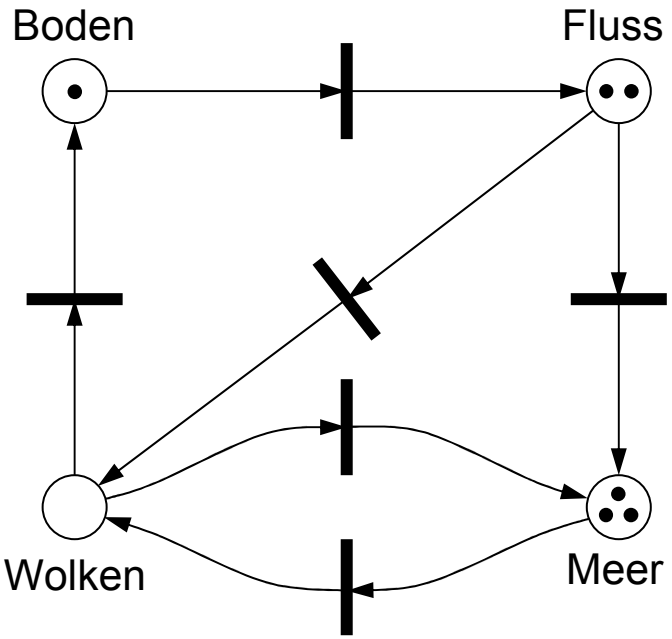
Transitionen

- Ferner können Stellen Marken enthalten:



Petri-Netze (1)

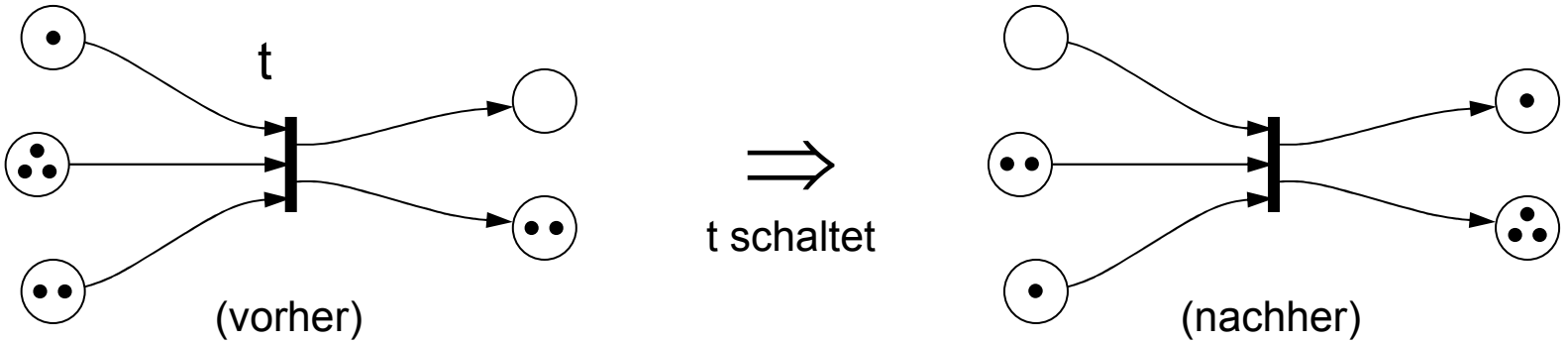
- Beispiel: Kreislauf des Wassers

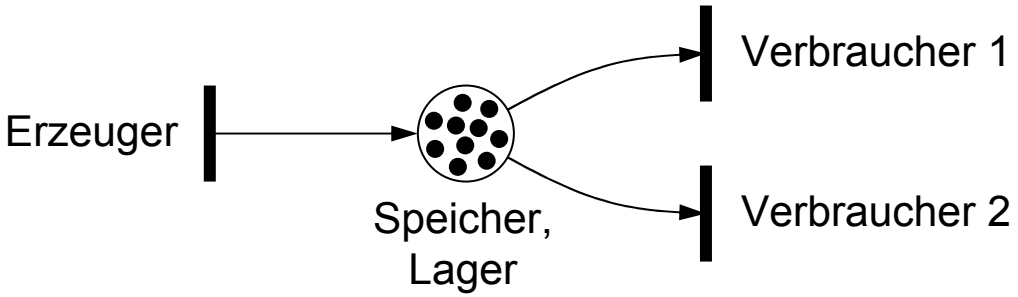


Petri-Netze (2)

- Die Transitionen können „schalten“ (oder „feuern“) unter der Voraussetzung, das alle Vorgängerstellen mindestens 1 Marke haben.

- Schalten bedeutet: Von jeder Vorgängerstelle wird eine Marke abgezogen und auf jede Nachfolgerstelle wird eine (zusätzliche) Marke gelegt.



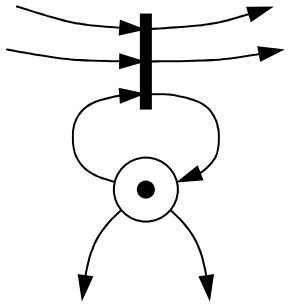


Petri-Netze (3)

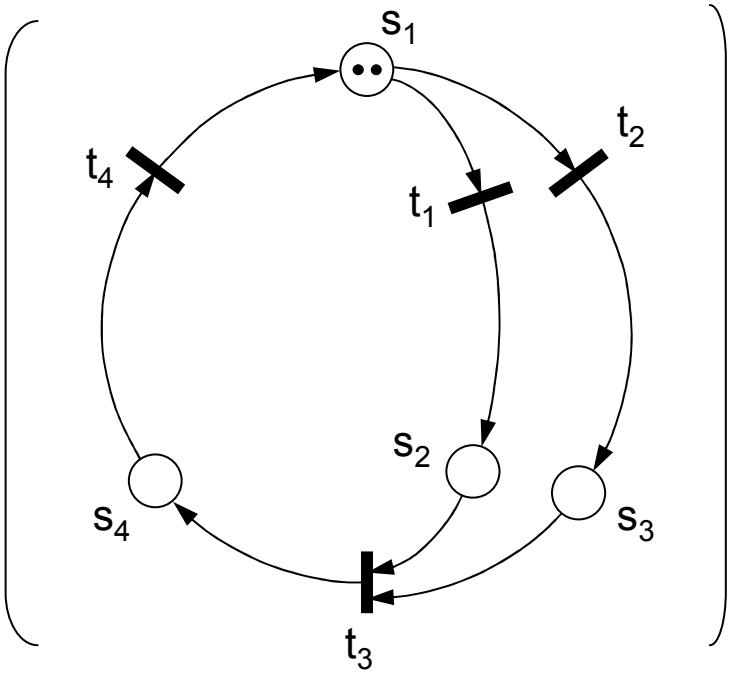
- Formal:
  - $M : s \rightarrow N_0$, Marken-Belegung
  - $t \in T$ kann schalten, wenn für alle Vorgänger $(s,t) \in E$ gilt: $M(s) \geq 1$
  - Nachdem t geschaltet hat, erhält man M‘ mit:

$$M'(s) = \begin{cases} M(s) + 1, & s \text{ ist Nachfolger, aber nicht Vorgänger von } t \\ M(s) - 1, & s \text{ ist Vorgänger, aber nicht Nachfolger von } t \\ M(s), & \text{sonst} \end{cases}$$

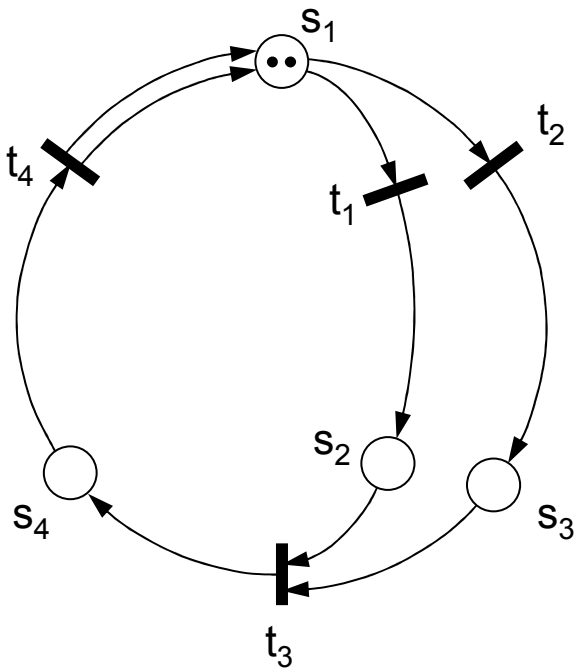
  - „sonst“ beinhaltet auch den Fall



Petri-Netze (4)



(auch Mehrfachkanten sind erlaubt)

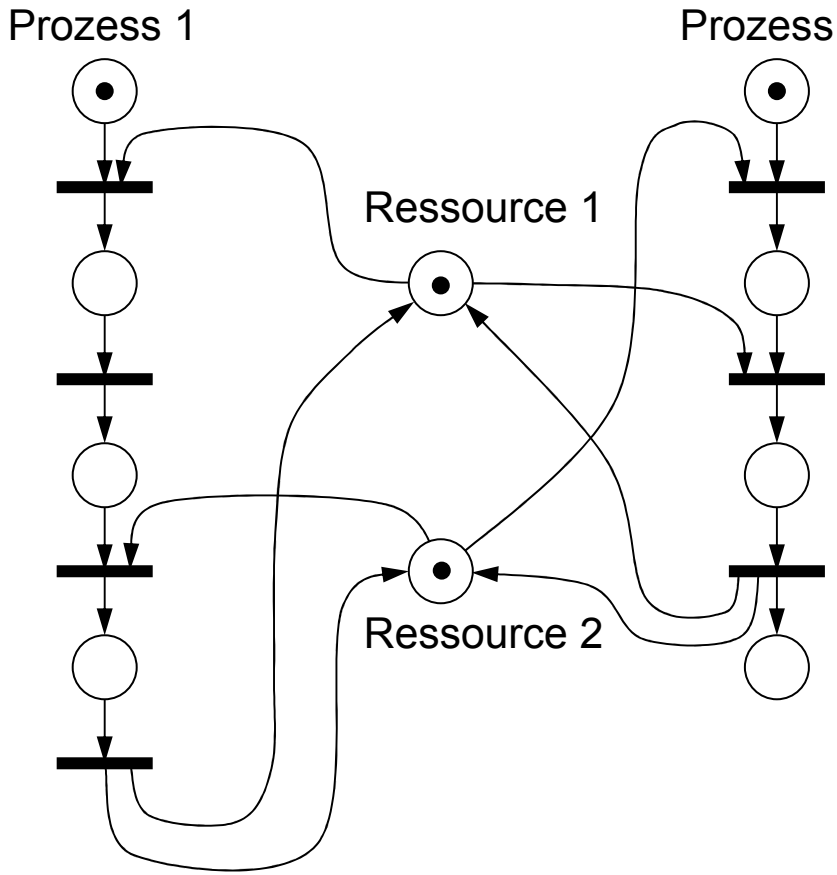


Aber: t_1, t_2

Jetzt kann keine Transition mehr schalten
(deadlock-Situation, Systemverklemmung)

Petri-Netze (5)

➢ Beispiel: Betriebssystem



z.B. Drucker

Petri-Netze (6)

- Notation: t_1, t_3, t_5 schalten: