# Graphen und Algorithmen

## Vorlesung #4: Maximale Flüsse

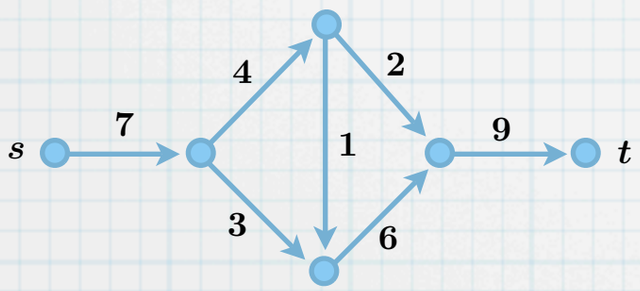
Dr. Armin Fügenschuh
Technische Universität Darmstadt
WS 2007/2008

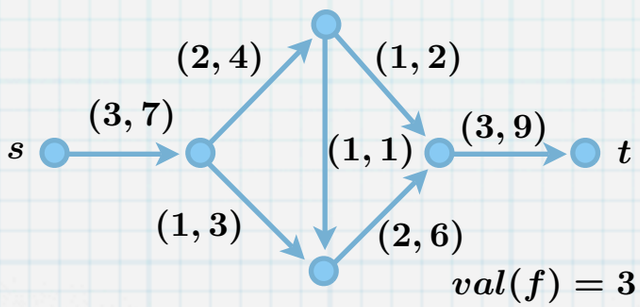
# Übersicht

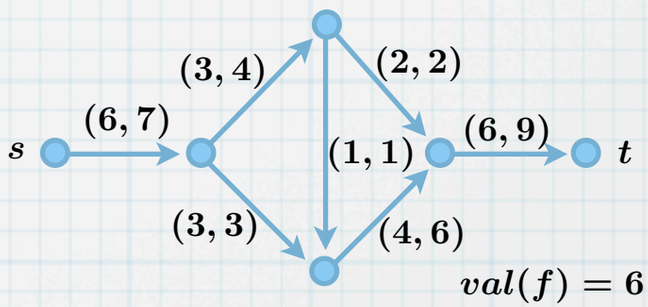
* Definition des Maximalfluss- und des Minimalschnitt-Problems
* Anwendungen
* Residualgraphen und augmentierende Wege
* Der Maximalfluss-Minimalschnitt-Satz
* Algorithmus von Ford und Fulkerson
* Eine Variante von Edmonds und Karp
* Eine weitere Variante von Dinits
* Algorithmus von Goldberg und Tarjan

# Das Problem des maximalen Fluss

* **Definition 1**:
  Sei $D = (V, A)$ ein zusammenhängender Digraph mit **Kapazitäten** $u : A \to \mathbb{R}_+$ und zwei ausgezeichneten Knoten $s, t \in V, s \neq t$, **Quelle** und **Senke** genannt. Das Quadrupel $(D, u, s, t)$ wird als **Flussnetz** bezeichnet.
* **Definition 2**:
  Sei $(D, u, s, t)$ ein Flussnetz. Ein **Fluss** ist eine Abbildung $f : A \to \mathbb{R}_+$ mit $f(a) \leq u(a)\ \forall\, a \in A$.
  $f$ erfüllt die **Flusserhaltung** am Knoten $v \in V$, wenn $\sum_{w:(w,v)\in A} f(w, v) = \sum_{w:(v,w)\in A} f(v, w)$.
  Eine **Zirkulation** ist ein Fluss, der an jedem Knoten $v \in V$ die Flusserhaltung erfüllt.
  Ein $s$–$t$–**Fluss** erfüllt die Flusserhaltung an jedem Knoten $v \in V \backslash \{s, t\}$.
  Der **Wert** eines $s$–$t$–Flusses ist definiert als $val(f) := \sum_{v:(s,v)\in A} f(s, v) - \sum_{v:(v,s)\in A} f(v, s)$.
* Beispiel:




* **Definition 3**:
  Gegeben sei ein Flussnetz $(D, u, s, t)$. Die Aufgabe, einen $s$–$t$–Fluss mit maximalem Wert zu finden, wird als **Maximalfluss-Problem** (engl. **max-flow problem**) bezeichnet.

# Existenz von Optimallösungen des Maximalfluss-Problems

* **Satz 4**:
  Sei $(D, u, s, t)$ ein Flussnetz. Dann hat das Maximalfluss-Problem eine optimale Lösung.
* Beweis:

  Jeder Fluss $f$ kann als Vektor $f = (f_{ij}) \in \mathbb{R}_+^{A(D)}$ gesehen werden.

  Sei $F$ die Menge der zulässigen Flussvektoren. Diese ist beschrieben durch

  $$0 \leq f_{i,j} \leq u_{i,j} \qquad \forall\,(i,j) \in A(D)$$

  $$\sum_{w:(v,w)\in A(D)} f_{v,w} = \sum_{w:(w,v)\in A(D)} f_{w,v} \qquad \forall\, v \in V(D)\backslash\{s,t\}$$

  $F$ ist eine beschränkte, abgeschlossene Teilmenge des $\mathbb{R}^{A(D)}$.

  Also ist $F$ kompakt (Satz von Weierstraß).

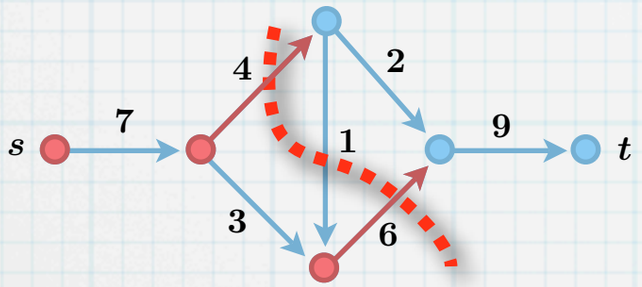
  Ferner ist $F \neq \emptyset$, da zumindest $0 \in F$.

  Die Funktion $val : \mathbb{R}^{A(D)} \to \mathbb{R}$ ist stetig.

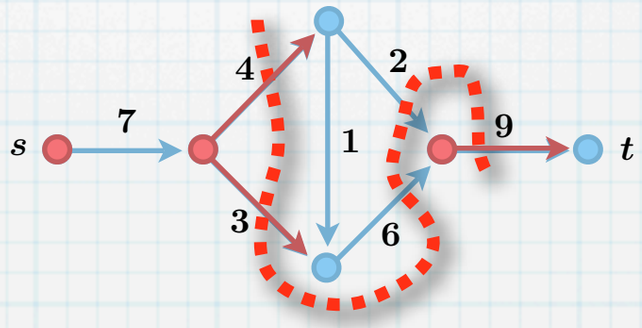
  Es folgt, dass $val$ auf $F$ ihr Maximum annimmt, was gleichbedeutend mit der Existenz einer optimal Lösung ist.

# Das Problem des minimalen Schnitts

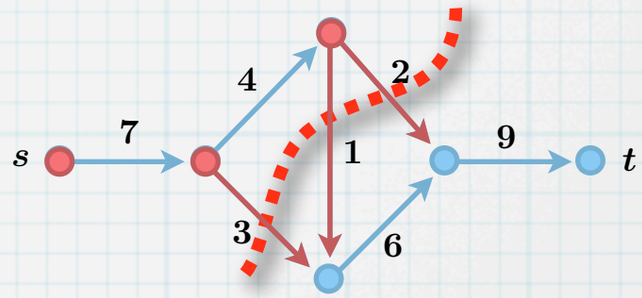
* **Definition 5**:
  Sei $(D, u, s, t)$ ein Flussnetz und $X \subset V$ eine Teilmenge der Knoten mit $s \in X, t \in V(D) \backslash X$.
  Dann wird die Teilmenge der Bögen $\{(i, j) \in A : i \in X, j \notin X\}$ als $s$–$t$–**Schnitt** bezeichnet.
  Die **Schnittkapazität** $u(X)$ ist die Summe der Bogenkapazitäten des $s$–$t$–Schnitts.
* **Definition 6**:
  Sei $(D, u, s, t)$ ein Flussnetz. Die Aufgabe, einen $s$–$t$–Schnitt mit minimalem Gewicht zu finden, wird als **Minimalschnitt-Problem** (engl. **min-cut problem**) bezeichnet.
* Beispiel:




$u(X) = 10$

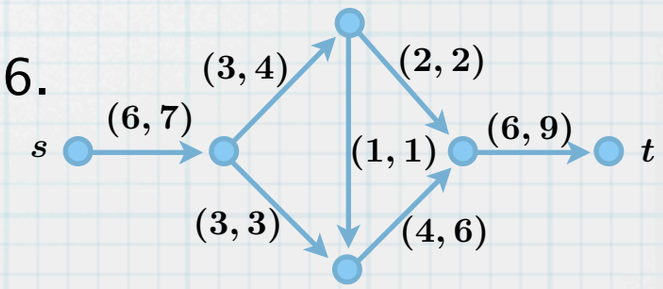



$u(X) = 16$




$u(X) = 6$

* Wir haben einen Fluss mit Wert 6 und einen Schnitt mit Kapazität 6.
* Im Folgenden werden wir zeigen, dass dieses kein Zufall ist, sondern auf einem fundamentalen Zusammenhang, dem max-flow min-cut Theorem, beruht: in einem solchen Fall beweist es, dass wir zu beiden Problemen Optimallösungen gefunden haben.




$val(f) = 6$

# Zusammenhang von Flüssen und Schnitten

* **Lemma 7**:
  Sei $(D, u, s, t)$ ein Flussnetz und $X \subset V$ eine Teilmenge der Knoten mit $s \in X, t \in V(D) \backslash X$.
  Sei $f$ ein $s$–$t$–Fluss. Dann gelten:

  (a) $val(f) = \sum_{\substack{(w,v) \in A \\ w \in X, v \notin X}} f(w,v) - \sum_{\substack{(v,w) \in A \\ v \notin X, w \in X}} f(v,w)$.

  (b) $val(f) \leq u(X)$.

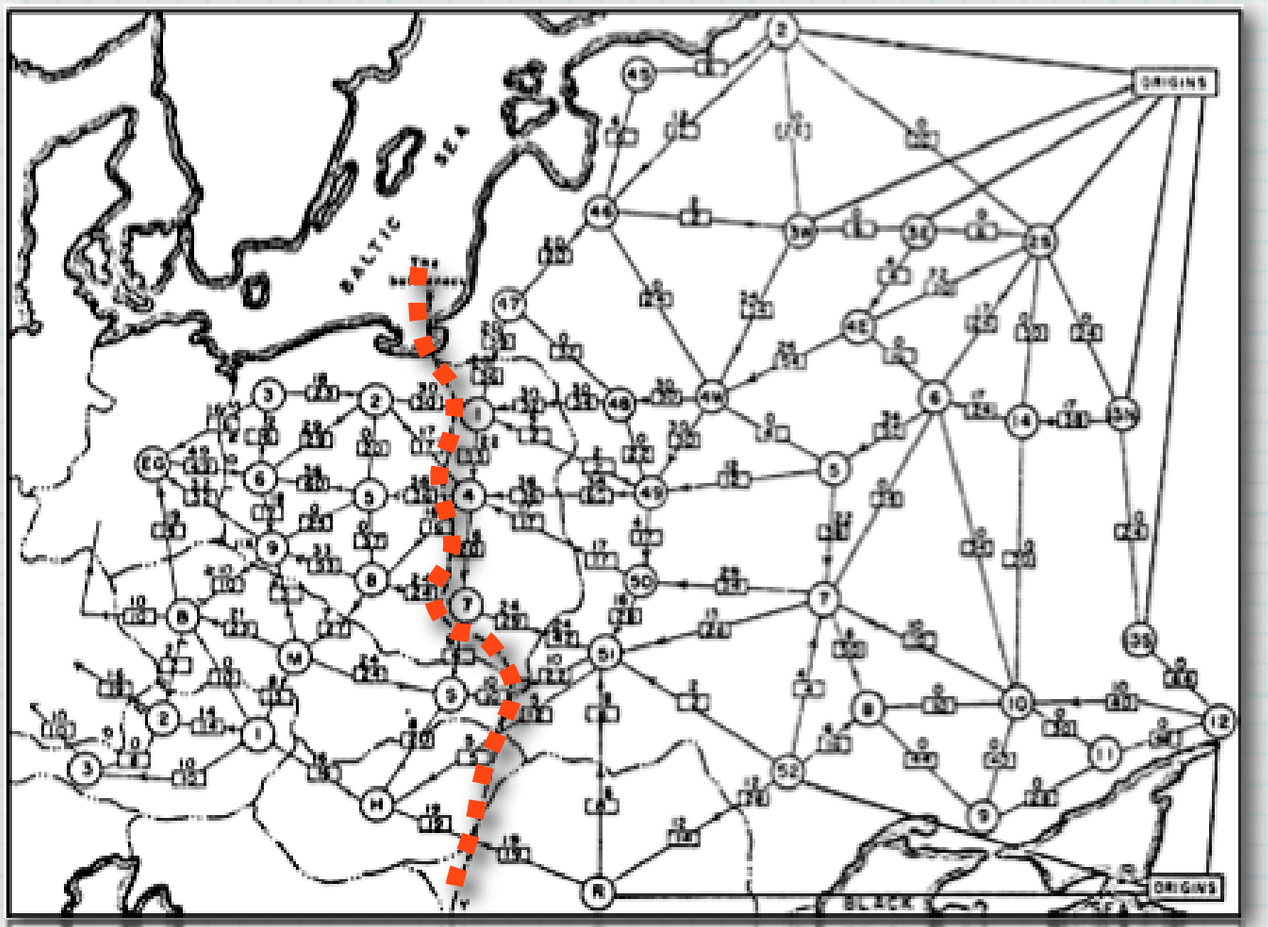
* Bemerkungen zu (a): Für $X := \{s\}$ entspricht dies genau der Definition von $val(f)$.
  Für $X := V(D) \backslash \{t\}$ zeigt es, dass der Wert des Flusses am „anderen Ende“ ankommt.

* Beweis (von Lemma 7):

| | |
|---|---|
| (Definition) | $val(f) = \sum_{v:(s,v) \in A} f(s,v) - \sum_{v:(v,s) \in A} f(v,s)$ |
| (Flusserhaltung) | $= \sum_{w \in X} \left( \sum_{v:(w,v) \in A} f(w,v) - \sum_{v:(v,w) \in A} f(v,w) \right)$ |
| (entferne doppelte Bögen) | $= \sum_{\substack{(w,v) \in A \\ w \in X, v \notin X}} f(w,v) - \sum_{\substack{(v,w) \in A \\ v \notin X, w \in X}} f(v,w)$ |
| ($f \geq 0$) | $\leq \sum_{\substack{(w,v) \in A \\ w \in X, v \notin X}} f(w,v)$ |
| ($f \leq u$) | $\leq \sum_{\substack{(w,v) \in A \\ w \in X, v \notin X}} u(w,v)$ |
| (Definition) | $= u(X)$ |

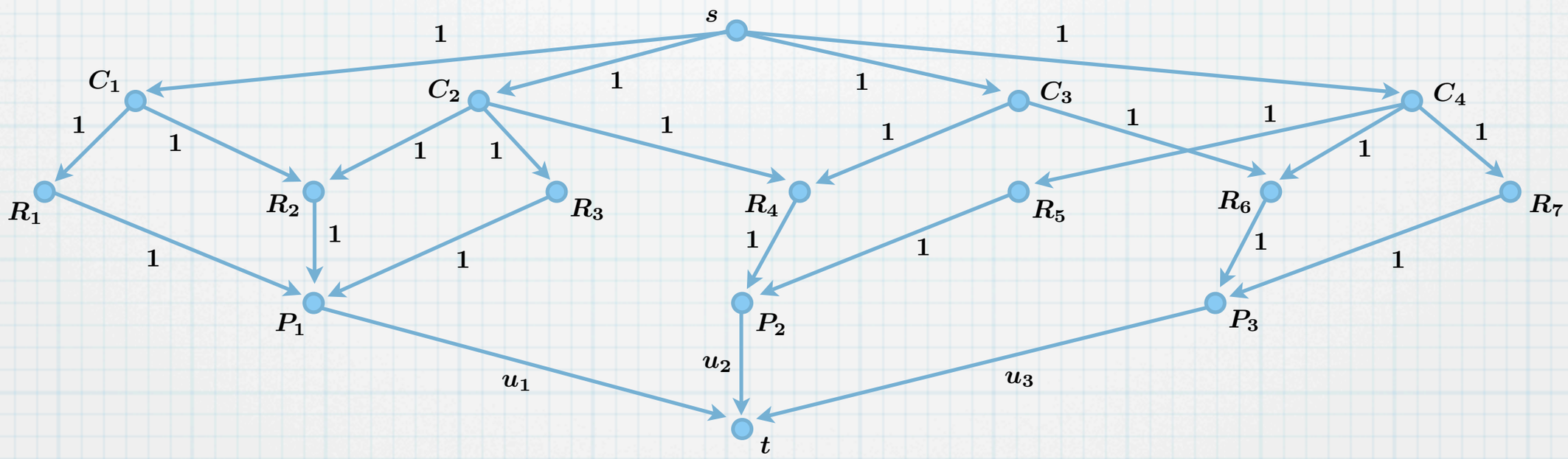
# Anwendungen (1): Transportkapazität auf Schienennetzen

* Harris-Ross-Report (1954), geheimgehalten bis 1999
* Schienennetz mit 44 Knoten und 105 Bögen
* Problem des Warschauer Pakts: Transport von maximal vielen Gütern nach Westen (max-flow)
* Problem der „Westens“: eben dieses zu verhindern (min-cut)




# Anwendung (2): Auswahlprobleme

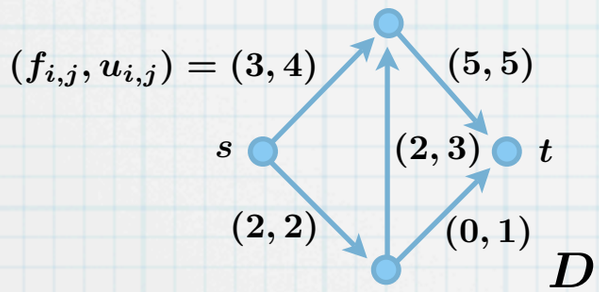
- Eine Stadt hat $r$ Einwohner $R_1, \ldots, R_r$, $q$ Vereine $C_1, \ldots, C_q$ und $p$ Parteien $P_1, \ldots, P_p$.
- Jeder Einwohner ist in mindestens einem Verein und in genau einer Partei.
- Jeder Verein stellt ein Mitglied auf als Abgeordneten in der Stadtversammlung.
- Die Gesamtzahl je Partei $P_k$ in der Versammlung ist begrenzt auf $u_k$ Abgeordnete.
- Gibt es eine Versammlung, die dieser Bedingung genügt?
- Modellierung als maximaler Fluss.
- Beispiel:

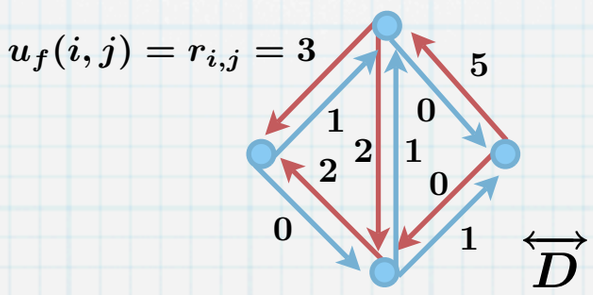


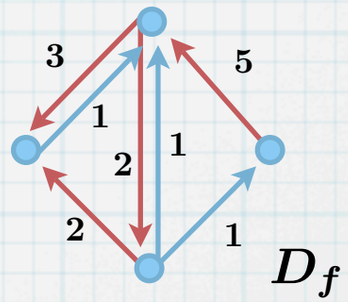

- Wenn der Wert des Flusses gleich $q$ ist, dann gibt es eine solche Versammlung, andernfalls nicht. (Bemerke, dass jeder Fluss mit Wert $q$ einer zulässigen Versammlung entspricht, und jede zulässige Versammlung als Fluss mit Wert $q$ gesehen werden kann.)

# Der Residualgraph

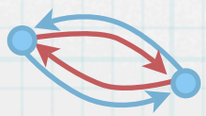
* **Definition 8**:
  Sei $D = (V, A)$ ein Digraph ohne gegenläufige Bögen. Für $a := (i, j) \in A$ sei $\overleftarrow{a} := (j, i)$ der **Rückwärtsbogen** von $a$ und $\overleftarrow{A} := \{\overleftarrow{a} : a \in A\}$ die Menge aller Rückwärtsbögen.
  Setze $\overleftrightarrow{A} := A \cup \overleftarrow{A}$ und $\overleftrightarrow{D} := (V, \overleftrightarrow{A})$. (Bemerke, dass $\overleftarrow{\overleftarrow{a}} = a$.)
  Sei $(D, u, s, t)$ ein Flussnetz und $f$ ein Fluss. Die **Restkapazität** (auch **Residualkapazität** oder **Residuum**) ist eine Abbildung $u_f : \overleftrightarrow{A} \to \mathbb{R}_+$, definiert durch $u_f(a) := u(a) - f(a)$ und $u_f(\overleftarrow{a}) := f(a)$ für alle Bögen $a \in A$. (Wir schreiben auch $r(a)$ anstelle von $u_f(a)$).
  Der **Residualgraph** ist definiert als $D_f := (V, \{a \in \overleftrightarrow{A} : u_f(a) > 0\})$.
* Beispiel:



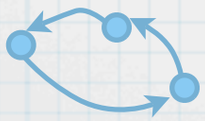



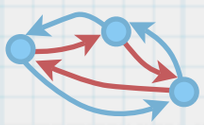




* Bemerkung: Wir betrachten hier (und im Folgenden) nur Digraphen ohne gegenläufige Bögen. Digraphen mit antiparallelen Bögen würden im Residualgraphen u.U. Multibögen zur Folge haben:

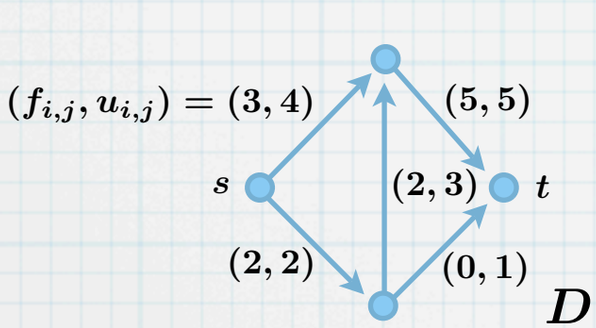

Diese Einschränkung ist jedoch kein Verlust an Allgemeinheit, da man gegenläufige Bögen durch Hinzufügen weiterer Knoten „unschädlich“ machen kann:

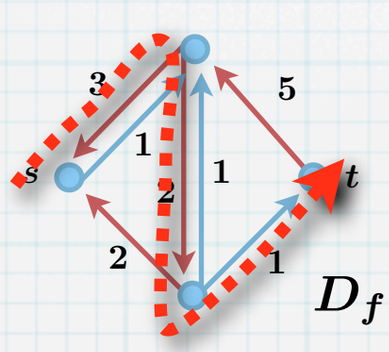


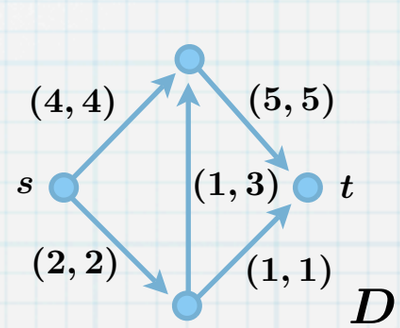



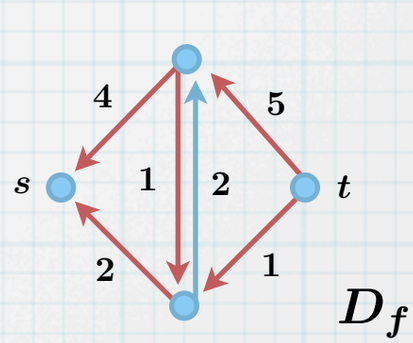
# Zunehmende Wege (augmenting paths)

* **Definition 9**:
  Sei $(D, u, s, t)$ ein Flussnetz, $f$ ein Fluss, $P$ ein (ungerichteter) Weg (oder ein Kreis) in $D_f$ und $\gamma > 0$ eine reelle Zahl. Unter der **Zunahme von $f$ entlang $P$ um $\gamma$** verstehen wir folgende Operation für alle Bögen $a$ des Pfades $P$:
  * Ist $a \in A$, dann erhöhe $f(a)$ um $\gamma$.
  * Ist $a \in \overleftarrow{A}$ (also $\overleftarrow{a} \in A$), dann erniedrige $f(\overleftarrow{a})$ um $\gamma$.
* **Definition 10**:
  Ein **$f$-augmentierender Weg** ist ein (vorwärtsgerichteter) $s$–$t$–Weg im Residualgraphen $D_f$.
* Beispiel:



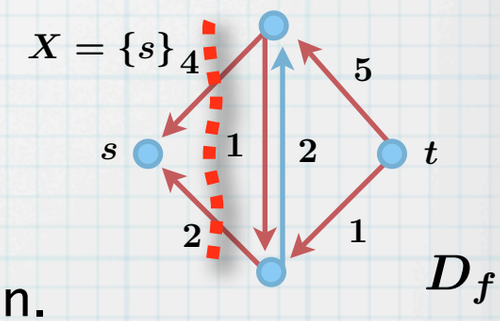
# Der Satz vom zunehmenden Weg

* **Satz 11** (augmenting path theorem):
  Ein Fluss $f$ auf einem Flussnetz $(D, u, s, t)$ ist genau dann maximal, wenn es keinen bezüglich $f$ zunehmenden Weg gibt.
* Beweis:
  * ($\Rightarrow$) Sei $f$ ein maximaler Fluss. Angenommen, es gibt einen bzgl. $f$ zunehmenden Weg $P$.
    Sei $\gamma := \min_{a \in P} u_f(a)$. Nach Definition des zunehmenden Weges ist $\gamma > 0$.
    Augmentiere $f$ entlang $P$ um $\gamma$. Sei $f'$ der augmentierte Fluss.
    Dann hat $f'$ den Wert $val(f') = val(f) + \gamma > val(f)$, im Widerspruch zur Maximalität.
  * ($\Leftarrow$) Angenommen, es gibt keinen bzgl. $f$ zunehmenden Weg $P$.
    Das heißt, es gibt im Residualgraphen $D_f$ keinen Weg von $s$ nach $t$.
    Sei $X$ die Menge aller Punkte $v \in V(D)$, für die es einen Weg von $s$ nach $v$ in $D_f$ gibt.
    Dann ist $s \in X$ und $t \notin X$.
    Also definiert $X$ einen $s$–$t$–Schnitt $S := \{(i,j) \in A : i \in X, j \notin X\}$.
    Es gibt keine Originalbögen in $D_f$, die aus $X$ herauszeigen.
    Also sind diese Bögen $a \in S$ gesättigt, d.h. $f(a) = u(a)$.
    Es gibt auch keine Rückwärtsbögen $\overleftarrow{a} \in D_f$, die aus $X$ herauszeigen.
    Also gilt für den jew. zugehörigen Originalbogen $a$, der in $X$ hineinzeigt, $f(a) = 0$.
    Betrachte nun die Abschätzung aus dem Beweis von Lemma 6:

$$val(f) = \ldots = \sum_{\substack{(w,v)\in A \\ w\in X, v\notin X}} f(w,v) - \sum_{\substack{(v,w)\in A \\ v\notin X, w\in X}} f(v,w) = \sum_{\substack{(w,v)\in A \\ w\in X, v\notin X}} f(w,v)$$

$$= \sum_{\substack{(w,v)\in A \\ w\in X, v\notin X}} u(w,v) = u(X)$$

    Somit ist $f$ maximal.



$D_f$

# Das Max-Flow Min-Cut Theorem und ganzzahlige Flüsse

* **Satz 12** (Ford, Fulkerson 1956; Elias, Feinstein, Shannon 1956):
  Der maximale Wert eines $s$–$t$–Flusses auf einem Flussnetz $(D, u, s, t)$ stimmt mit der minimalen Kapazität eines $s$–$t$–Schnitts auf diesem Netz überein.
* Beweis:

  Die Existenz von maximalen Flüssen wurde in Satz 4 gezeigt.

  Ein Fluss $f$ ist genau dann maximal, wenn die Menge $X$ aller von $s$ aus auf einem zunehmenden Weg erreichbaren Knoten $t$ nicht enthält (Beweis von Satz 11).

  In diesem Fall gilt $val(f) = u(X)$, siehe ebenfalls Beweis von Satz 11.
* **Satz 13**:
  Sei $(D, u, s, t)$ ein Flussnetz mit ganzzahligen Kapazitäten $u_{i,j} \in \mathbb{Z}$ für alle $(i, j) \in A(D)$.
  Dann gibt es einen ganzzahligen maximalen Fluss.
* Beweis:

  Durch $f_{i,j} := 0$ für alle $(i, j) \in A(D)$ wird ein ganzzahliger zulässiger Fluss definiert.

  Wenn dieser Fluss nicht maximal ist, gibt es einen $f$–augmentierenden Weg $P$.

  Sei $\gamma := \min_{a \in P} u_f(a)$. Nach Definition des zunehmenden Weges ist $\gamma > 0$.

  Da $u_f(i, j) \in \mathbb{Z}$ für alle $(i, j) \in A(D)$, ist auch $\gamma \in \mathbb{Z}$. Insbesondere ist $\gamma \geq 1$.

  Wir augmentieren $f$ entlang $P$ um $\gamma$ und erhalten so einen Fluss $f^{(1)}$ mit $val(f^{(1)}) = \gamma$.

  Dieses Verfahren wird nun wiederholt, bis es keinen $f$–zunehmenden Weg mehr gibt.

  Dieses ist nach endlich vielen Schritten der Fall, da der Flusswert stets um eine positive ganze Zahl erhöht wird, und die Kapazität eines (minimalen oder anderen) Schnitts eine obere Schranke darstellt.

  Nach Satz 11 ist der so konstruierte Fluss maximal.

# Algorithmus von Ford-Fulkerson

* Eingabe: Flussnetz $(D, u, s, t)$
* Ausgabe: ein $s$–$t$–Fluss $f$ mit maximalem Wert

(1) **algorithm** fordFulkerson
(2) setze $f(a) := 0$ für alle $a \in A$
(3) **while** es gibt einen $f$–augmentierenden Weg $P$ in $D_f$ **do**
(4) $\gamma := \min_{a \in P} u_f(a)$
(5) augmentiere $f$ entlang $P$ um $\gamma$
(6) **end while**
(7) **end algorithm**

* Bemerkungen:
  * Dieser Algorithmus funktioniert nur für ganzzahlige oder rationale Bogenkapazitäten.
    Der Fall rationaler Kapazitäten wird auf den ganzzahligen zurückgeführt durch Multiplikation mit dem Hauptnenner.
    Der Aufbau des Graphen $D_f$ ist in $O(|A|)$.
    Die Verbesserung beträgt (mind.) 1 in jeder Runde.
    Es gibt $F := \max\{val(f) : f \text{ ein Fluss}\}$ viele Runden.
    Im Fall ganzzahliger Kapazitäten kann das Verfahren (bei schlechter Wahl der augmentierenden Wege) exponentiell viele Schritte benötigen, $O(|A| \cdot F)$.
    Das Verfahren ist also nicht effizient.
  * Für irrationale Kapazitäten kann er versagen (bei schlechter Wahl der augmentierenden Wege): Das Verfahren terminiert u.U. nicht und es konvergiert nicht gegen Optimallösung.
    Das Verfahren ist in diesem Fall also nicht einmal korrekt.

# Algorithmus von Edmonds-Karp

* Eingabe: Flussnetz $(D, u, s, t)$
* Ausgabe: ein $s$–$t$–Fluss $f$ mit maximalem Wert

(1) **algorithm** edmondsKarp
(2) setze $f(a) := 0$ für alle $a \in A$
(3) **while** es gibt einen $f$–augmentierenden Weg in $D_f$ **do**
(4) $P :=$ $f$–augmentierender Weg in $D_f$ mit größtem Wert für $\gamma := \min_{a \in P} u_f(a)$
(5) augmentiere $f$ entlang $P$ um $\gamma$
(6) **end while**
(7) **end algorithm**

* Bemerkung: Die Suche nach einem Weg in Schritt (4), dessen minimales Bogengewicht maximal ist, erfolgt durch die Konstruktion eines maximalen gerichteten Spannbaums in $D_f$ mit Startknoten $s$.

  Dazu kann z.B. eine Variante von Jarniks Spannbaum–Algorithmus verwendet werden.

  Es wird ein gerichteter Baum $T$ aufgebaut durch schrittweises Hinzufügen des Bogens mit der jeweils höchsten Kapazität, solange, bis $T$ einen gerichteten Weg von $s$ nach $t$ enthält.

  Die Komplexität dafür ist in $O(|V|^2)$ (siehe Laufzeitbeweis des Algorithmus von Jarnik).

# Zur Laufzeit des Edmonds-Karp-Algorithmus

* **Satz 14**:
  Die Edmonds–Karp–Variante des Algorithmus von Ford–Fulkerson ist für ganzzahlige Flussnetze effizient; genauer gesagt in $O(|V|^2 \cdot |A| \cdot \log F)$, wobei $F$ der Wert eines maximalen Flusses ist.
* Beweis:

  Sei $f$ irgendein Fluss im Flussnetz $(D, u, s, t)$.

  Sei $P$ ein $f$–augmentierender Weg in $D_f$ mit größtem Wert für $\gamma$ (Schritt 4).

  Sei $a \in P$ derjenige Bogen mit $\gamma = u_f(a)$ (anschaulich: der „Flaschenhals“).

  Sei $X$ die Menge der Knoten $v$, die von $s$ aus erreichbar ist über Bögen in $D$, deren jeweilige Kapazität größer als $\gamma$ ist.

  Ferner ist $t \notin X$. Also definiert $X$ einen $s$–$t$–Schnitt.

  Jede Kante, die über diesen Schnitt verläuft (in Richtung $t$) hat eine Kapazität $\leq \gamma$.

  Damit ist $u(X) \leq \gamma \cdot |A|$ (triviale Abschätzung).

  Anderseits ist $u(X) \geq val(f)$ (Lemma 7b).

  Beides zusammen bedeutet, dass $\gamma \geq val(f)/|A|$ gilt.

  Wir augmentieren $f$ entlang $P$ um $\gamma$:

  $$val(f) \mapsto val(f) + \gamma \geq val(f) + val(f)/|A| = val(f) \cdot (1 + 1/|A|).$$

  In jeder Iteration (Schritt 3) wächst der Fluss um mindestens einen Faktor von $(1 + 1/|A|)$.

  Nach $|A| \cdot \log F$ Iterationen gilt für den Zuwachs dann:

  $$(1 + 1/|A|)^{|A| \cdot \log F} > 2^{\log F} = F.$$

  Also ist der Zuwachs dann größer als der maximale Flusswert, der Algorithmus terminiert.

  Die Laufzeit ergibt sich aus $O(|A| \cdot \log F)$ Iterationen in (3) und $O(|V|^2)$ für (4).

# Algorithmus von Dinits

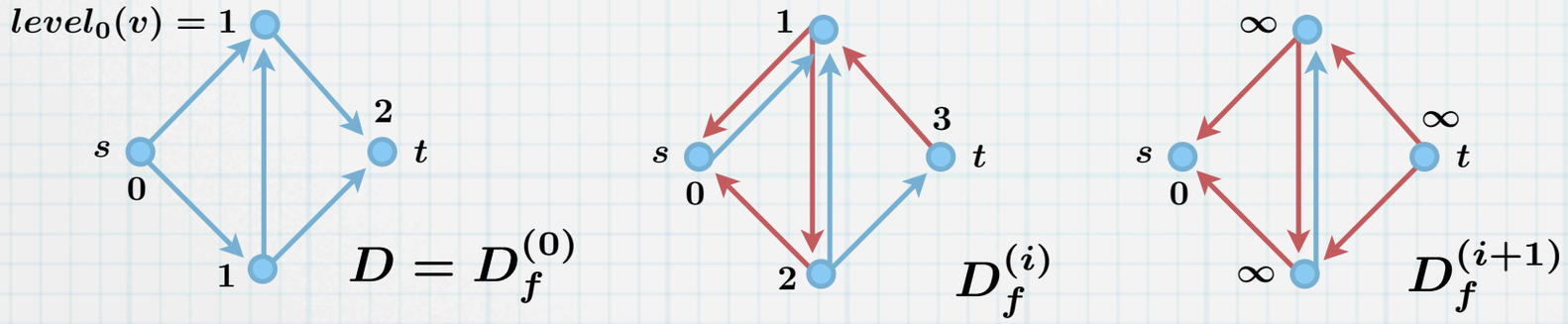
* Eingabe: Flussnetz $(D, u, s, t)$
* Ausgabe: ein $s$–$t$–Fluss $f$ mit maximalem Wert

(1) **algorithm** dinits
(2) setze $f(a) := 0$ für alle $a \in A$
(3) **while** es gibt einen $f$–augmentierenden Weg in $D_f$ **do**
(4) $P :=$ minimaler (bzgl. Kantenanzahl) $f$–augmentierender Weg in $D_f$
(5) $\gamma := \min_{a \in P} u_f(a)$
(6) augmentiere $f$ entlang $P$ um $\gamma$
(7) **end while**
(8) **end algorithm**

* Bemerkung: Die Suche nach einem minimalen Weg in Schritt (4) erfolgt als eine Breitensuche.

# Laufzeit-Analyse des Dinits-Algorithmus

* **Satz 14**:
  Die Dinits-Variante des Algorithmus von Ford-Fulkerson ist für beliebige Flussnetze effizient und in $O(|V| \cdot |A|^2)$.
* Der Beweis von Satz 14 beruht auf zwei Beobachtungen über die Entwicklung des Residualgraphen während der Ausführung des Algorithmus.
  Sei $f^{(i)}$ der Fluss nach $i$ Verbesserungsschritten und sei $D_f^{(i)}$ der zugehörige Residualgraph.
  (Es ist $f_{i,j}^{(0)} = 0$ für alle $(i,j) \in A$ und $D_f^{(0)} = D$.)
  Für jeden Knoten $v \in V$ sei $level_i(v)$ der Level von $v$ in einer Breitensuche in $D_f^{(i)}$ beginnend in $s$ (mit anderen Worten: die Länge des kürzesten Weges von $s$ nach $v$ in $D_f^{(i)}$).
* Beispiel:



# Laufzeit-Analyse (Forts.)

* **Lemma 15**:
  Der Level eines Knotens nimmt während der Iterationen monoton zu: $level_{i+1}(v) \geq level_i(v)$.
* Beweis (per Induktion über den Level-Wert $k$):
  Für Level-Wert $k = 0$ klar, da $level_i(v) = 0$ genau für $v = s$ gilt (für alle $i$).
  Sei die Aussage wahr für alle $i$ und ein festes Level $k$. Wir schließen auf $k+1$ (für alle $i$).
  Sei $v \neq s$ und $P = (s, \ldots, w, v)$ ein kürzester $s$–$v$–Weg in $D_f^{(i+1)}$ der Länge $k+1$.
  Da $P$ ein kürzester Weg ist, gilt $level_{i+1}(v) = level_{i+1}(w) + 1$.
  $P$ beinhaltet einen kürzesten $s$–$w$–Weg der Länge $k$.
  Also gilt nach Induktionsvoraussetzung: $level_{i+1}(w) \geq level_i(w)$.
  1. Fall, $(w, v)$ ist Bogen in $D_f^{(i)}$.
     Dann ist $level_i(v) \leq level_i(w) + 1$.
  2. Fall, $(w, v) \notin D_f^{(i)}$.
     Also ist entweder der Fluss $f^{(i)}$ auf Bogen $(v, w)$ leer oder auf Bogen $(w, v)$ gesättigt.
     Da aber $(w, v) \in D_f^{(i+1)}$, ist entweder $f^{(i+1)}$ auf $(v, w)$ nicht mehr leer oder auf $(w, v)$ nicht mehr gesättigt.
     Es gilt $(v, w) \in D_f^{(i)}$ und Bogen $(v, w)$ ist im $f^{(i)}$–augmentierenden Weg enthalten.
     Also ist $(v, w)$ auf einem kürzesten $s$–$t$–Weg in $D_f^{(i)}$.
     Daher gilt: $level_i(v) = level_i(w) - 1 \leq level_i(w) + 1$.
  In beiden Fällen gilt: $level_{i+1}(v) = level_{i+1}(w) + 1 \geq level_i(w) + 1 \geq level_i(v)$.
  Bemerke: Für Knoten $v \neq s$, zu denen es keinen $s$–$v$–Weg gibt, gilt $level_{i+1}(v) = \infty \geq level_i(v)$.

# Laufzeit-Analyse (Forts.)

* **Lemma 16**:
  Sei $a \in A$. Während der Ausführung des Algorithmus ist Bogen $a$ bzw. $\overleftarrow{a}$ höchstens $|V|/2$ Mal nicht im Residualgraphen $D_f^{(i)}$.
* Beweis:
  Angenommen, Bogen $(w, v)$ ist in den Residualgraphen $D_f^{(i)}$ und $D_f^{(j+1)}$ für $i < j$, aber in keinem der Residualgraphen $D_f^{(i+1)}, \ldots, D_f^{(j)}$.
  Das bedeutet, dass $(w, v)$ im $i$–ten zunehmenden Weg ist.
  Dann ist $level_i(v) = level_i(w) + 1$.
  Ferner bedeutet es, dass $(v, w)$ im $j$–ten zunehmenden Weg ist.
  Also ist $level_j(w) = level_j(v) + 1$.
  Nach Lemma 15 gilt: $level_j(w) = level_j(v) + 1 \geq level_i(v) + 1 = level_i(w) + 2$.
  Interpretation: Die Distanz $level_j(w) - level_i(w)$ hat sich zwischen Verschwinden und Wiedererscheinen von Bogen $(w, v)$ um mindestens 2 erhöht.
  Der Level–Wert ist entweder kleiner als $|V|$ oder unendlich ($\infty$).
  Somit kann $(w, v)$ höchstens $|V|/2$ Mal verschwinden.
* Beweis (von Satz 14):
  Alle $|A|$ Bögen können dann zusammen $|A| \cdot |V|/2$ Mal verschwinden.
  In jeder Iteration verschwindet mindestens ein Bogen.
  Also terminiert der Algorithmus nach $|A| \cdot |V|/2$ Iterationen (Schritt 3).
  Der Aufwand für jede dieser Iterationen ist in $O(|A|)$ für die Breitensuche (Schritt 4).
  Somit ist die Komplexität in $O(|V| \cdot |A|^2)$.
* **Korrolar 17**:
  Für planare Graphen ist Dinits Algorithmus in $O(|V|^3)$, für dichte Graphen in $O(|V|^5)$.

# Auf dem Weg zu einem schnelleren Verfahren: Präflüsse

* Wir haben gezeigt (Satz 11), dass eine Abbildung $f : A \to \mathbb{R}_+$ genau dann ein maximaler $s$–$t$–Fluss im Flussnetz $(D, u, s, t)$ ist, wenn sie die folgenden Eigenschaften erfüllt:

1. $0 \leq f_{i,j} \leq u_{i,j}$ $\qquad \forall\, (i,j) \in A$
2. $\sum_{w:(w,v)\in A} f_{w,v} = \sum_{w:(v,w)\in A} f_{v,w}$ $\qquad \forall\, v \in V \backslash \{s,t\}$
3. es gibt keinen $f$–zunehmenden Weg.

* Der Ford-Fulkerson-Algorithmus (und seine Varianten) erfüllt während der Ausführung stets die ersten beiden Eigenschaften, und terminierten, wenn die dritte erfüllt ist.
* Im Folgenden stellen wir den Algorithmus von Goldberg-Tarjan vor, der die erste und dritte Eigenschaft während der Ausführung erfüllt, und terminiert, sobald auch die zweite erfüllt ist.

Während der Ausführung ist $f$ kein Fluss. Um zu erklären, was $f$ dann ist, führen wir folgende Sprechweise ein.

* **Definition 18**:
Sei $(D, u, s, t)$ ein Flussnetz. Eine Abbildung $f : A \to \mathbb{R}_+$ mit $f(a) \leq u(a)$ für alle $a \in A$ und

$$ex_f(v) := \sum_{w:(w,v)\in A} f_{w,v} - \sum_{w:(v,w)\in A} f_{v,w} \geq 0 \qquad \forall\, v \in V \backslash \{s\}$$

wird $s$–$t$–**Präfluss** genannt ($ex$ = excess, **Überschuss**).
Ein Knoten $v \in V \backslash \{s,t\}$ heißt **aktiv**, falls $ex_f(v) > 0$ gilt.

# Distanzabschätzungen

* **Definition 19**:
  Eine Funktion $\psi_f := \psi : V \to \mathbb{N}$ mit $\psi(t) = 0, \psi(s) = n$ und $\psi(v) \leq \psi(w) + 1 \; \forall \, (v,w) \in A(D_f)$ wird als **Distanzabschätzung** bezeichnet.
  Ein Bogen $a = (v,w) \in \overleftrightarrow{A}$ ist **zulässig**, wenn $a \in A(D_f)$ und $\psi(v) = \psi(w) + 1$.
* **Lemma 20**:
  Sei $\psi$ eine Distanzabschätzung und $v \in V \backslash \{s\}$. Dann ist $\psi(v)$ eine Abschätzung für die Distanz (Anzahl Bögen in einem kürzesten $v$–$t$–Weg) in $D_f$.
* Beweis (durch Induktion über die Kantenzahl):
  Sei $dist(v,t)$ die Länge eines kürzesten $v$–$t$–Weges in $D_f$.
  Für $v = t$ ist $\psi(t) = 0 = dist(t,t)$.
  Für $(v,t) \in D_f$ ist $\psi(v) \leq \psi(t) + 1 = 0 + 1 = 1 = dist(v,t)$.
  Sei die Aussage gezeigt für alle kürzesten $v$–$t$–Wege der Länge $k$.
  Sei $P$ ein kürzester $v$–$t$–Weg mit $k+1$ Bögen. Sei $(v,w)$ der erste Bogen dieses Weges.
  Dann gilt:

$$\begin{aligned} \psi(v) &\leq \psi(w) + 1 && \text{(Definition)} \\ &\leq dist(w,t) + 1 && \text{(Induktionsvoraussetzung)} \\ &\leq dist(v,t) && (P \text{ ist kürzester Weg}). \end{aligned}$$

# Die Unterroutinen „push“ und „relabel“

* **Definition 21**:
  Die **Zunahme** (**Augmentierung**) eines Präflusses $f$ **entlang eines Bogens** $a \underset{\leftarrow}{\in} A(D_f)$ **um** $\gamma$ ist die Operation $f(a) \mapsto f(a) + \gamma$ für $a \in A$ und $f(a) \mapsto f(a) - \gamma$ für $a \in \overleftarrow{A}$.
* „push“ und „relabel“ sind zwei wichtige Subroutinen im Goldberg-Tarjan-Algorithmus.
* push($a$) schiebt den Überschuss (bzw. Teile davon) eines Knotens über den angegebene Bogen, ohne die Kapazitätsgrenze von $a$ zu überschreiten bzw. negativen Fluss zu erzeugen.

(1) **sub** push($a$)
(2) $\gamma := \min\{ex_f(v), u_f(a)\}$, wobei $a = (v, w)$
(3) augmentiere $f$ entlang $a$ um $\gamma$
(4) **end sub**

* Bemerke: Beim Anwenden von push($a$) bleibt $f$ ein Präfluss.
* relabel($v$) passt die Distanzabschätzung für Knoten $v$ an den aktuellen Residualgraphen an.

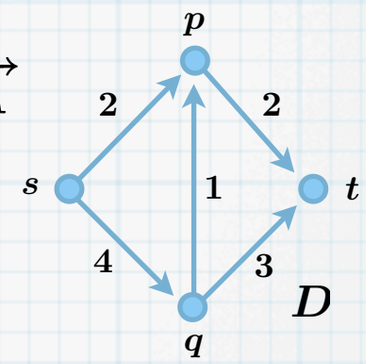
(1) **sub relabel**($v$)
(2) $\psi(v) := \min\{\psi(w) + 1 : a = (v, w) \in A(D_f)\}$
(3) **end sub**

* Bemerke:
  Für alle $v \in V \backslash \{s, t\}, (v, w) \in A(D_f)$ ist $\psi(v) := \min\{\psi(u) + 1 : (v, u) \in A(D_f)\} \leq \psi(w) + 1$.
  Also bleibt $\psi$ auch nach Anwenden von relabel($v$) eine Distanzabschätzung.

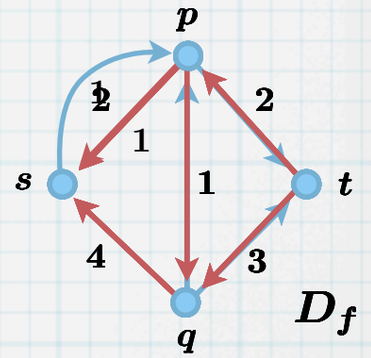
# Der Algorithmus von Goldberg-Tarjan

* Eingabe: Flussnetz $(D, u, s, t)$, Adjazenzlisten $(A_v)_{v \in V}$ für $\overleftrightarrow{A}$
* Ausgabe: maximaler $s$–$t$–Fluss $f$

```
(1)  algorithm goldbergTarjan
(2)    f_{s,v} := u_{s,v} ∀(s,v) ∈ A, f_{v,w} := 0 ∀(v,w) ∈ A, s ≠ v
(3)    ψ(s) := n, ψ(v) := 0 ∀v ∈ V\{s}
(4)    nextArc(v) := 1 für alle v ∈ V
(5)    repeat
(6)      Q := {v ∈ V\{s,t} : ex_f(v) > 0}
(7)      for alle v ∈ Q do
(8)        repeat
(9)          w := A_v(nextArc(v))
(10)         if (v,w) ist zulässig then push(v,w)
(11)         else
(12)           if (v,w) ist letzter Bogen in A_v then
(13)             relabel(v)
(14)             nextArc(v) := 1
(15)             goto (19)
(16)           else nextArc(v) := nextArc(v) + 1 end if
(17)         end if
(18)       until ex_f(v) = 0
(19)     end for
(20)   until Q = ∅
(21) end algorithm
```




| *Endknoten* | 1 | 2 | 3 |
|---|---|---|---|
| $A_s(\cdot)$ | $p$ | $q$ | |
| $A_p(\cdot)$ | $s$ | $q$ | $t$ |
| $A_q(\cdot)$ | $s$ | $p$ | $t$ |
| $A_t(\cdot)$ | $p$ | $q$ | |




$Q = \{p, q\}$

$v = p$

$w = s$

$(p, s) \in A(D_f), 0 = \psi(p) \neq \psi(s) + 1 = 5$

$\psi(p) := \min\{5, 1\} = 1$

| *Bogen* | $sp$ | $sq$ | $qp$ | $pt$ | $qt$ |
|---|---|---|---|---|---|
| $f(\cdot)$ | 2 | 4 | 0 | 0 | 0 |

| *Knoten* | $p$ | $q$ | $t$ |
|---|---|---|---|
| $ex_f(\cdot)$ | 2 | 4 | 0 |

| *Knoten* | $s$ | $p$ | $q$ | $t$ |
|---|---|---|---|---|
| $\psi(\cdot)$ | 4 | 0 | 0 | 0 |

| *Knoten* | $s$ | $p$ | $q$ | $t$ |
|---|---|---|---|---|
| $nextArc(\cdot)$ | 1 | 1 | 1 | 1 |

# Das Erhaltungslemma

* **Lemma 22**:
  Während der Ausführung des Goldberg-Tarjan Algorithmus ist $f$ stets ein Präfluss und $\psi$ stets eine Distanzabschätzung.
* Beweis:
  Es bleibt nur zu zeigen, dass $\psi$ auch nach push eine Distanzabschätzung ist.
  Nach Anwenden von push($a$) auf $a = (v, w)$ kommt ggfs. $\overleftarrow{a} = (w, v)$ neu zu $D_f$ hinzu.
  Da $a$ ein zulässiger Bogen ist, gilt $\psi(v) = \psi(w) + 1$.
  Also: $\psi(w) = \psi(v) - 1 \leq \psi(v) + 1$, was zu beweisen war.
* **Lemma 23**:
  relabel($v$) in Schritt (13) wird nur aufgerufen, wenn $v$ aktiv und kein Bogen $(v, w)$ zulässig ist.
* Beweis:
  Ein Knoten $v \in Q$ bleibt so lange aktiv, bis er durch die Schritte (8)-(18) läuft.
  relabel($v$) wird daher nur aufgerufen, wenn $v$ auch aktiv ist.
  Zu zeigen: für alle $a = (v, w) \in A(D_f)$ gilt $\psi(w) \geq \psi(v)$, d.h. kein Bogen ist zulässig.
  Seit dem letzten Aufruf von relabel($v$) wurde die gesamte Adjazenzliste $A_v$ durchlaufen.
  Zu irgendeinem Zeitpunkt zeigte $nextArc(v)$ also auf $a$.
  Der Zeiger $nextArc(v)$ wird nur erhöht, wenn die Kante, auf die er zeigt, unzulässig ist.
  Das bedeutet entweder $\psi(v) < \psi(w) + 1$ oder $a \notin A(D_f)$.
  Im ersten Fall bleibt $\psi(w) \geq \psi(v)$ so lange, bis relabel($v$) aufgerufen, denn $\psi(w)$ kann bis dahin nur weiter wachsen.
  Im zweiten Fall muss später ein push($\overleftarrow{a}$) erfolgt sein, da jetzt $a \in A(D_f)$ ist.
  Zum Zeitpunkt des Aufrufs von push($\overleftarrow{a}$) gilt bereits $\psi(w) = \psi(v) + 1$.
  Später kann $\psi(w)$ durch weitere relabel($w$)-Aufrufe nur erhöht werden.

# Erreichbare Knoten im Residualgraphen

* **Lemma 24**:
  Sei $f$ ein $s$–$t$–Präfluss und $\psi = \psi_f$ eine Distanzabschätzung. Dann gilt:
  (a) $s$ ist in $D_f$ von jedem aktiven Knoten $v$ aus erreichbar.
  (b) $t$ ist in $D_f$ nicht erreichbar von $s$.
* Beweis:
  (a) Sei $v$ ein aktiver Knoten, d.h. $ex_f(v) > 0$.
  Sei $R$ die Menge der von $v$ aus in $D_f$ erreichbaren Knoten.
  Betrachte einen Bogen $(i,j) \in A$ mit $i \notin R, j \in R$.
  Angenommen, $f_{i,j} > 0$. Dann ist $(j,i) \in D_f$, also $i \in R$. Widerspruch. Also ist $f_{i,j} = 0$.
  Es gilt: $ex_f(w) = \sum_{x:(x,w)\in A} f_{x,w} - \sum_{x:(w,x)\in A} f_{w,x} \quad \forall\, w \in R$

$$\Rightarrow \quad \sum_{w\in R} ex_f(w) = \sum_{w\in R} \left( \sum_{x:(x,w)\in A} f_{x,w} - \sum_{x:(w,x)\in A} f_{w,x} \right)$$

$$= \sum_{\substack{(x,w)\in A \\ x\notin R, w\in R}} f_{x,w} - \sum_{\substack{(w,x)\in A \\ w\in R, x\notin R}} f_{w,x} \leq 0.$$

  Da aber $v \in R$ ein aktiver Knoten ist, also $ex_f(v) > 0$, muss es noch (mind.) einen Knoten $w_0 \in R$ geben mit $ex_f(w_0) < 0$.
  Weil $f$ ein $s$–$t$–Präfluss ist, kann es sich dabei nur um $w_0 = s$ handeln.
  (b) Angenommen, es gibt einen $s$–$t$–Weg in $D_f$, z.B. $P = (v_0, v_1, \ldots, v_k)$ mit $v_0 = s, v_k = t$.
  Bemerke, dass $k \leq n-1$, wobei $n := |V|$.
  Da $\psi$ eine Distanzabschätzung bzgl. $f$ ist, gilt $\psi(v_i) \leq \psi(v_{i+1}) + 1$ für $i = 0, \ldots, k-1$.
  Also ist $\psi(s) \leq \psi(v_1) + 1 \leq \psi(v_2) + 2 \leq \ldots \leq \psi(t) + k$.
  Nun ist $\psi(s) = n$ und $\psi(t) = 0$, also heißt das: $n \leq 0 + k = k$. Widerspruch.

# Korrektheit von Goldberg-Tarjan

* **Satz 25**:
  Wenn der Algorithmus von Goldberg-Tarjan terminiert, dann ist $f$ ein maximaler $s$-$t$-Fluss.
* Beweis:

  Wenn der Algorithmus terminiert, gibt es keine aktiven Knoten mehr.

  Daher ist $f$ dann ein $s$-$t$-Fluss.

  Da es keinen $s$-$t$-Weg in $D_f$ gibt, kann $f$ nicht augmentiert werden.

  Somit ist $f$ maximal.

# Zur Anzahl der relabel-Aufrufe

* **Lemma 26**:
  (a) $\psi(v)$ nimmt niemals ab. Durch jeden Aufruf von relabel($v$) nimmt $\psi(v)$ zu.
  (b) Es ist stets $\psi(v) \leq 2n-1$ für alle Knoten $v \in V$, wobei $n := |V|$.
  (c) relabel wird höchstens $2n^2$ Mal aufgerufen.
* Beweis:
  (a) $\psi$ wird nur durch die Subroutine relabel verändert.
  relabel wird nur aufgerufen, wenn kein Bogen $(v,w)$ zulässig ist (Lemma 23).
  In diesem Fall ist $\psi(v) < \psi(w)+1$ für alle Bögen $(v,w) \in A(D_f)$.
  Also ist $\psi(v) < \min\{\psi(w)+1 : (v,w) \in A(D_f)\}$.
  Durch die Setzung $\psi(v) := \min\{\psi(w)+1 : (v,w) \in A(D_f)\}$ nimmt $\psi(v)$ zu.
  (b) $\psi(v)$ wird nur geändert, wenn $v$ aktiv ist (Lemma 23).
  Nach Lemma 24a) gibt es für aktive $v$ in $D_f$ einen $v$–$s$–Weg.
  Sei $P = (v_0, v_1, \ldots, v_k)$ mit $v_0 = v, v_k = s$ ein solcher Weg.
  Dann gilt: $\psi(v) = \psi(v_0) \leq \psi(v_1)+1 \leq \psi(v_2)+2 \leq \ldots \leq \psi(v_k)+k$
  $= \psi(s)+k \leq n+(n-1) = 2n-1.$
  (c) Bei jedem relabel–Aufruf nimmt $\psi(v)$ zu, siehe a).
  Jeder Knoten kann höchstens $2n-1$ Mal einen neuen Wert bekommen, siehe b).
  Es gibt $n$ Knoten, also maximal $(2n-1)n \leq 2n^2$ Aufrufe von relabel.

# Zur Anzahl der gesättigten Schübe

* **Definition 27**:
  Ist $u_f(a) = 0$ (also $f(a) = u(a)$ ) nach dem Aufruf von push($a$), dann sprechen wir von einem **gesättigten Schub**, andernfalls von einem **ungesättigten Schub.**
* **Lemma 28**:
  Die Anzahl der gesättigten Schübe ist $\leq m \cdot n$, wobei $m := |A|$ und $n := |V|$.
* Beweis:

  Sei $a = (v, w) \in A$ ein Bogen, der durch push($a$) gesättigt wird.
  Danach wird $a$ aus $D_f$ gelöscht und ggfs. kommt $\overleftarrow{a} = (w, v)$ neu zu $D_f$ hinzu.
  Vor dem Ausführen von push($a$) war $a$ zulässig. Also galt $\psi(v) = \psi(w) + 1$.
  Bevor $a$ wieder für einen Schub gewählt werden kann, muss folgendes geschehen:
  $\psi(w)$ muss um 2 erhöht werden.
  Dadurch ist $\psi(w) = \psi(v) + 1$, also ist $\overleftarrow{a}$ zulässig.
  Dann muss push($\overleftarrow{a}$) ausgeführt werden, damit $\overleftarrow{\overleftarrow{a}} = a$ wieder in $D_f$ erscheint.
  Danach muss $\psi(v)$ um 2 erhöht werden.
  Dadurch ist $\psi(v) = \psi(w) + 1$, also ist $a$ wieder zulässig.

Also: für 2 Aufrufe push($a$) muss $\psi(v)$ um 2 wachsen.
Nach Lemma 26b) geht das nur $\lfloor (2n - 1)/2 \rfloor < n$ Mal (je Bogen).
Insgesamt gibt es daher höchstens $m \cdot n$ gesättigte Schübe.

# Zur Anzahl der ungesättigten Schübe

* **Lemma 29**:
  Die Anzahl der ungesättigten Schübe ist höchstens $4n^3$.
* Beweis:

  In jeder inneren Iteration kann es max. einen unges. Schub pro Knoten $v \in Q$ geben.

  Denn: sei $a = (v, w)$ und push($a$) ungesättigt.

  Dann ist $\gamma = \min\{ex_f(v), u_f(a)\} = ex_f(v) < u_f(a)$.

  Somit gilt $ex_f(v) = 0$ für den Präfluss $f$ nach Ausführung von push($a$).

  Danach wird der nächste Knoten $v \in Q$ behandelt.

  Dieses wird $n$ Mal wiederholt.

  Bleibt zu zeigen: es gibt höchstens $4n^2$ innere Iterationen.

  Bemerke: Es gibt höchstens $2n^2$ innere Iterationen mit relabel.

  Ziel: Abschätzung der inneren Iterationen ohne relabel.

  Setze $\Psi := \max\{\psi(v) : v \text{ aktiv}\}$ und $\Psi := 0$, falls kein Knoten aktiv.

  Es gilt $\Psi = 0$, wenn Algorithmus terminiert und $\Psi > 0$ während der Ausführung.

  Während der Ausführung verändert sich $\Psi$ nicht-monoton.

  $\Psi$ kann nur wachsen, wenn $\psi(v)$ wächst für ein $v$ (durch relabel).

  Also ist das Gesamtwachstum von $\Psi$ nach Lemma 26c) beschränkt durch $2n^2$.

  Es bleibt nur noch zu zeigen, dass $\Psi$ in jeder Iteration ohne relabel um mindestens 1 reduziert wird, d.h. $\Psi_{vorher} - \Psi_{nachher} \geq 1$.

# Zur Anzahl der ungesättigten Schübe (Forts.)

* Beweis von Lemma 29 (Forts.):

  Behauptung: In jeder Iteration ohne relabel wird $\Psi$ um mindestens 1 reduziert, d.h. $\Psi_{vorher} - \Psi_{nachher} \geq 1$.

  1. Fall, es gibt „nachher“ noch aktive Knoten, z.B. Knoten $w$ mit $\psi(w) = \Psi_{nachher}$.

     1. Fall, $w$ war vorher nicht aktiv.

        Dann gibt es einen Knoten $v$ und es wurde push$(v, w)$ aufgerufen.

     2. Fall, $w$ war vorher bereits aktiv, d.h. $w \in Q$.

        Da wir eine Iteration ohne relabel betrachten, wird diese so lange wiederholt, bis der Überschuss komplett abgebaut wurde.

        Am Ende ist aber noch Überschuss vorhaden, also muss von einem anderen Knoten $v$ aus push$(v, w)$ aufgerufen worden sein.

     In jedem Fall war auch $v$ aktiv.

     Weil ein push$(v, w)$ erfolgte, war $(v, w)$ zulässig, d.h. $\psi(v) = \psi(w) + 1$.

     Somit gilt: $\Psi_{nachher} = \psi(w) = \psi(v) - 1 \leq \Psi_{vorher} - 1$.

  2. Fall, „nachher“ gibt es keine aktiven Knoten mehr, also $\Psi_{nachher} = 0$.

     Da es kein relabel gab, gab es push-Operationen, z.B. push$(v, w)$ für Bogen $(v, w)$.

     Weil $(v, w)$ zulässig ist, gilt $\psi(v) = \psi(w) + 1$, also
     $\Psi_{vorher} \geq \psi(v) = \psi(w) + 1 \geq 0 + 1 = \Psi_{nachher} + 1$.

# Laufzeitkomplexität von Goldberg-Tarjan

* **Satz 30**:
  Der Algorithmus von Goldberg–Tarjan bestimmt einen maximalen $s$–$t$–Fluss in $O(n^3)$.
* Beweis:
  Es gibt höchstens $2n^2$ relabel–Aufrufe.
  Jeder Aufruf ist linear in $O(n)$.
  Damit ist der Gesamtaufwand für relabel in $O(n^3)$.
  Es gibt $m \cdot n \leq n^3$ gesättigte push–Aufrufe und $4n^3$ ungesättigte.
  Jeder Aufruf von push ist konstant in $O(1)$.
  Damit ist der Gesamtaufwand für push in $O(n^3)$.
  Also ist die Komplexität des Goldberg–Tarjan–Verfahrens in $O(n^3)$.

# Literaturquellen


* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 6–8, Seite 166–293)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 8, Seite 287–320)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 3, Seite 37–90)
* T.H. Cormen, C.E. Leiserson, R.E. Rivest, C. Stein, **Introduction to Algorithms**, 2nd Edition, McGraw-Hill Book Company, Boston, 2001. (Kapitel 26, Seite 643–700)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana-Champaign, 2007. (Kapitel 15–16)
* D. Jungnickel: Graphen, **Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 6, Seite 201–262)
* B. Korte, J. Vygen: **Combinatorial Optimization – Theory and Algorithms**, 2nd Edition, Springer Verlag, Berlin, 2001. (Kapitel 8, Seite 153–184)
* R. Möhring, **Graphen und Netzwerkalgorithmen**, Vorlesungsskript, 2005. (Kapitel 5)
* A. Schrijver: **Combinatorial Optimization – Polyhedra and Efficiency**. Springer Verlag, Berlin, 2003. (Kapitel 10–11, Seite 148–176)