

Datenstrukturen und Algorithmen

Vorlesung 18: Maximaler Fluss (K26)

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

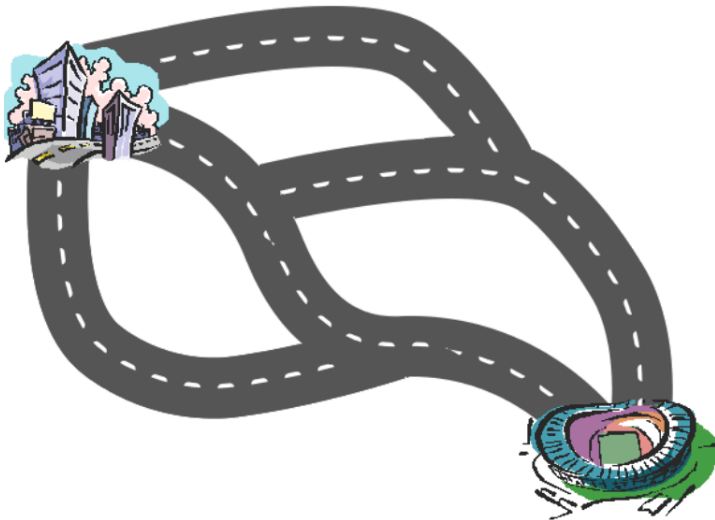
<https://moves.rwth-aachen.de/teaching/ss-18/dsal/>

29. Juni 2018

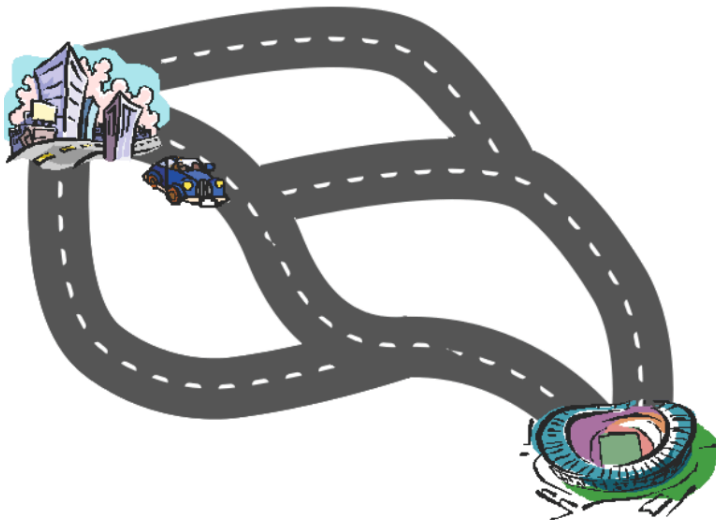
Übersicht

- 1 Flussnetzwerke
- 2 Ford-Fulkerson-Methode
 - Restnetzwerke
 - Algorithmus
 - Schnitte
- 3 Anwendungen und Erweiterungen
 - Edmonds-Karp-Algorithmus
 - Alternative Algorithmen

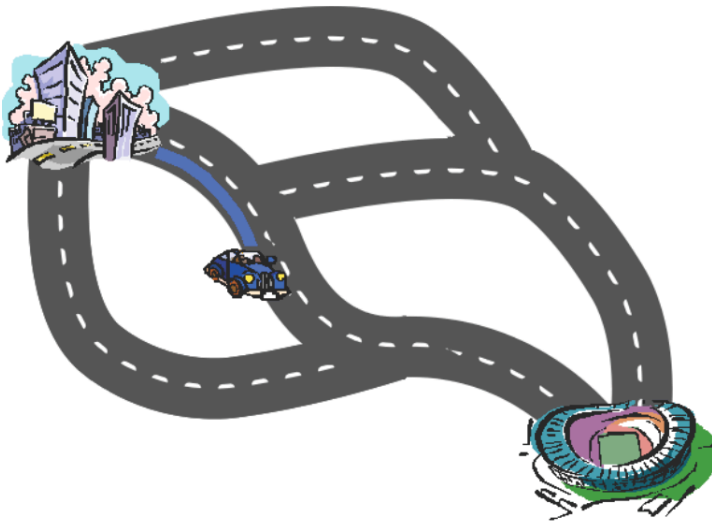
Graphenproblem: maximale Flüsse



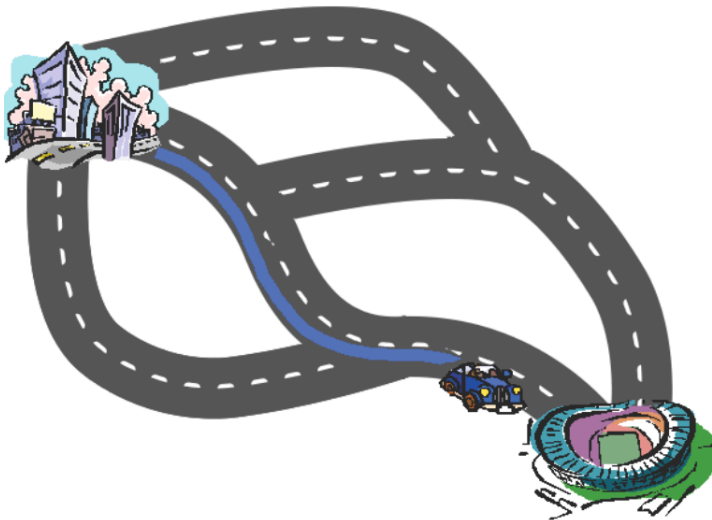
Graphenproblem: maximale Flüsse



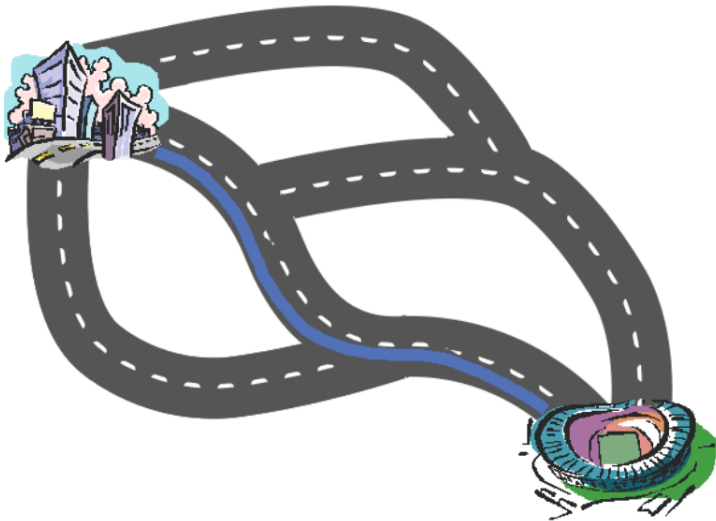
Graphenproblem: maximale Flüsse



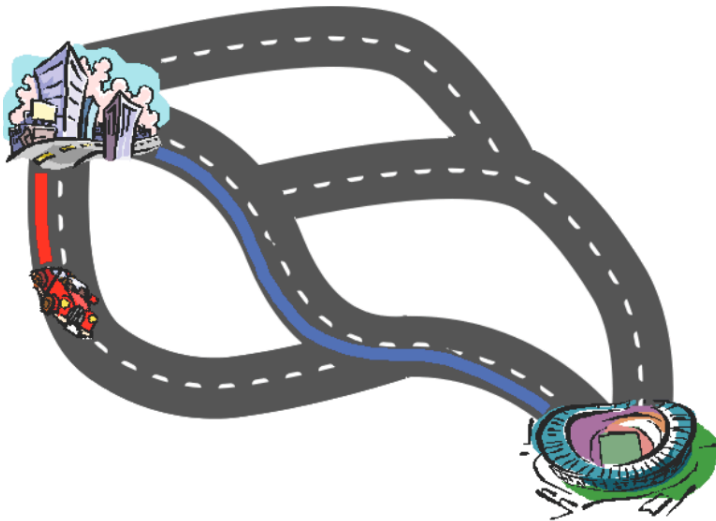
Graphenproblem: maximale Flüsse



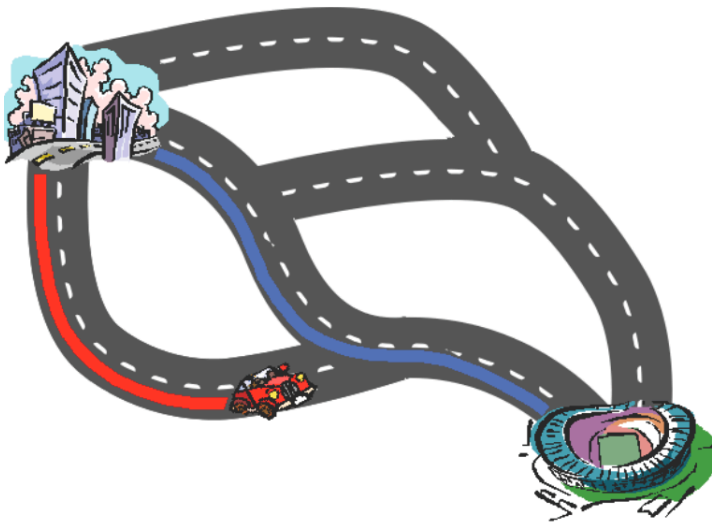
Graphenproblem: maximale Flüsse



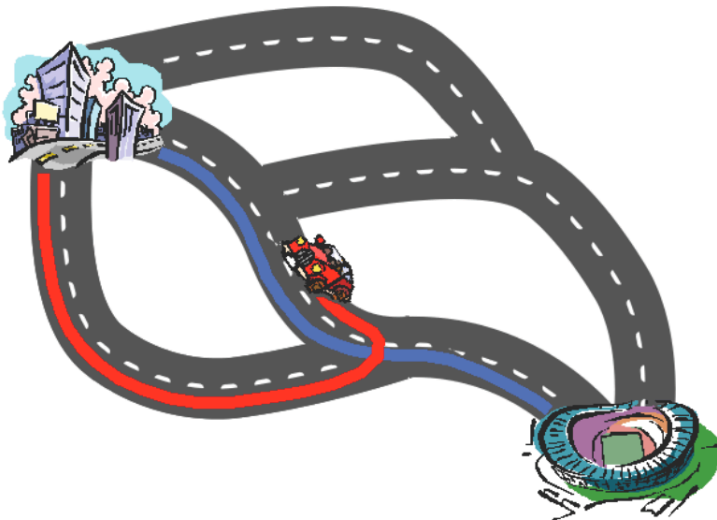
Graphenproblem: maximale Flüsse



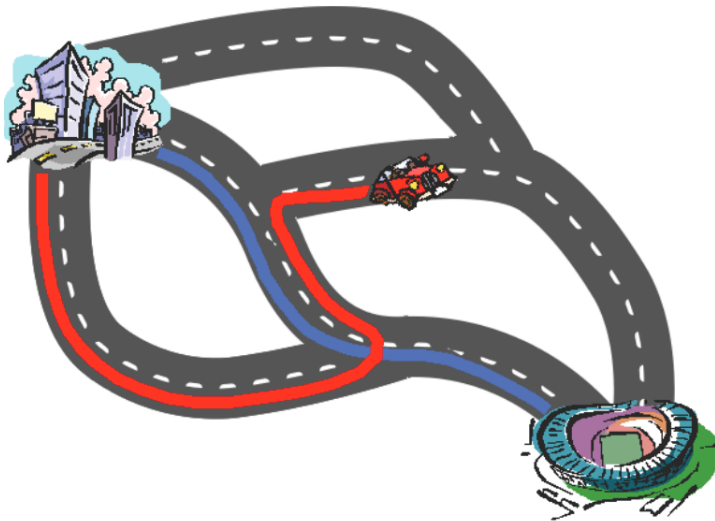
Graphenproblem: maximale Flüsse



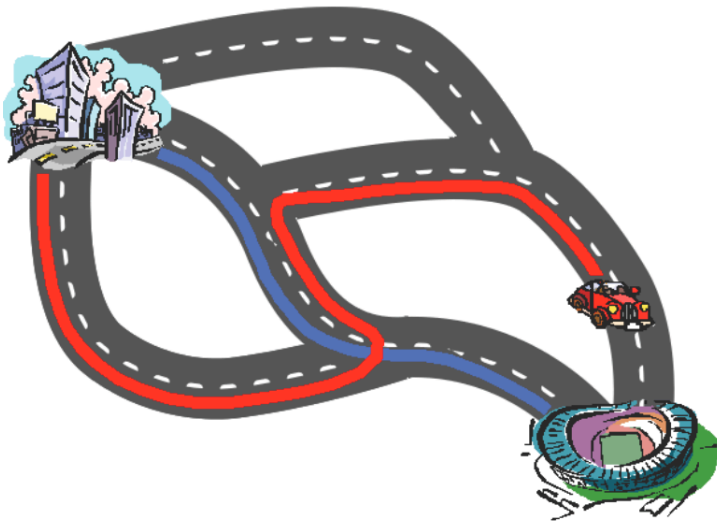
Graphenproblem: maximale Flüsse



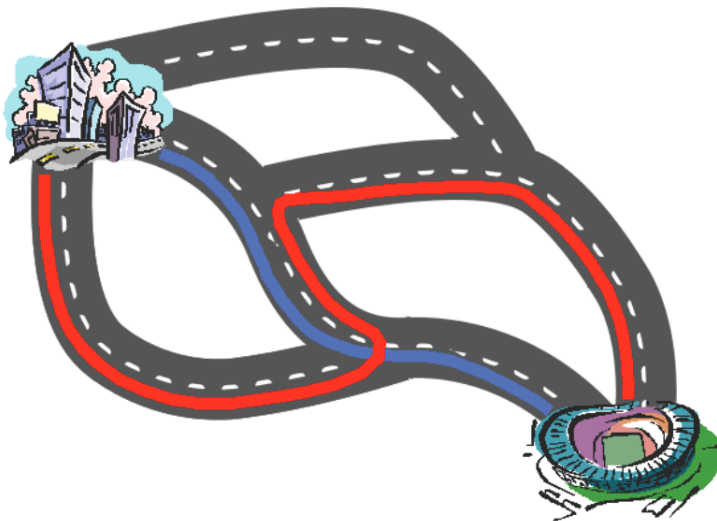
Graphenproblem: maximale Flüsse



Graphenproblem: maximale Flüsse



Graphenproblem: maximale Flüsse



Graphenproblem: maximale Flüsse

Beispiel (maximale Flüsse)

Eingabe:

1. Eine Straßenkarte, auf der die Kapazität der Straßen eingezeichnet ist,
2. eine Quelle, und
3. eine Senke.

Ausgabe: Die maximale Rate, mit der Material (= Zuschauer) von der Quelle bis zur Senke (= Stadion) transportiert werden kann, ohne die Kapazitätsbeschränkungen der Straßen zu verletzen.

Übersicht

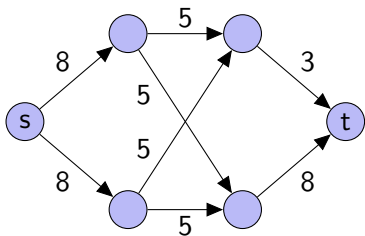
- 1 Flussnetzwerke
- 2 Ford-Fulkerson-Methode
 - Restnetzwerke
 - Algorithmus
 - Schnitte
- 3 Anwendungen und Erweiterungen
 - Edmonds-Karp-Algorithmus
 - Alternative Algorithmen

Flussnetzwerk

Flussnetzwerk

Ein **Flussnetzwerk** $G = (V, E, c)$ ist ein digraph (V, E) mit

- ▶ $c : V \times V \longrightarrow \mathbb{R}_{\geq 0}$ die **Kapazitätsfunktion** sodaß:
 - ▶ $(u, v) \in E$ dann $c(u, v) \geq 0$
 - ▶ $(u, v) \notin E$ dann $c(u, v) = 0$
- ▶ $s, t \in V$, die **Quelle** s und **Senke** t des Flussnetzwerkes
- ▶ Jeder Knoten $v \in V$ liegt auf einem Pfad von Quelle s zur Senke t

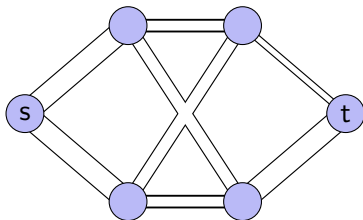


Flussnetzwerk

Flussnetzwerk

Ein **Flussnetzwerk** $G = (V, E, c)$ ist ein digraph (V, E) mit

- ▶ $c : V \times V \longrightarrow \mathbb{R}_{\geq 0}$ die **Kapazitätsfunktion** sodaß:
 - ▶ $(u, v) \in E$ dann $c(u, v) \geq 0$
 - ▶ $(u, v) \notin E$ dann $c(u, v) = 0$
- ▶ $s, t \in V$, die **Quelle** s und **Senke** t des Flussnetzwerkes
- ▶ Jeder Knoten $v \in V$ liegt auf einem Pfad von Quelle s zur Senke t



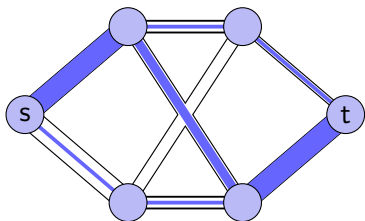
- ▶ An der Quelle wird produziert
- ▶ An der Senke wird verbraucht
- ▶ Kanten sind wie Wasserrohre
- ▶ Kapazität = maximale Durchsatzrate

Flussnetzwerk

Flussnetzwerk

Ein **Flussnetzwerk** $G = (V, E, c)$ ist ein digraph (V, E) mit

- ▶ $c : V \times V \longrightarrow \mathbb{R}_{\geq 0}$ die **Kapazitätsfunktion** sodaß:
 - ▶ $(u, v) \in E$ dann $c(u, v) \geq 0$
 - ▶ $(u, v) \notin E$ dann $c(u, v) = 0$
- ▶ $s, t \in V$, die **Quelle** s und **Senke** t des Flussnetzwerkes
- ▶ Jeder Knoten $v \in V$ liegt auf einem Pfad von Quelle s zur Senke t



- ▶ An der Quelle wird produziert
- ▶ An der Senke wird verbraucht
- ▶ Kanten sind wie Wasserrohre
- ▶ Kapazität = maximale Durchsatzrate

Fluss in einem Flussnetzwerk

Definition (Fluss)

Ein **Fluss** ist eine Funktion $f: V \times V \rightarrow \mathbb{R}$, mit folgenden Eigenschaften:

Beschränkung: Für $u, v \in V$ gilt $f(u, v) \leq c(u, v)$.

Asymmetrie: Für $u, v \in V$ gilt $f(u, v) = -f(v, u)$.

Flusserhaltung: Für $u \in V \setminus \{s, t\}$ gilt: $\sum_{v \in V} f(u, v) = 0$.

$f(u, v)$ ist der Fluss vom Knoten u zum Knoten v .

Fluss in einem Flussnetzwerk

Definition (Fluss)

Ein **Fluss** ist eine Funktion $f: V \times V \rightarrow \mathbb{R}$, mit folgenden Eigenschaften:

Beschränkung: Für $u, v \in V$ gilt $f(u, v) \leq c(u, v)$.

Asymmetrie: Für $u, v \in V$ gilt $f(u, v) = -f(v, u)$.

Flusserhaltung: Für $u \in V \setminus \{s, t\}$ gilt: $\sum_{v \in V} f(u, v) = 0$.

$f(u, v)$ ist der Fluss vom Knoten u zum Knoten v .

- ▶ Der Fluss zwischen zwei Knoten darf die Kapazitätsbeschränkung nicht überschreiten
- ▶ Der Fluss von u nach v ist der negative Wert des Flusses von v nach u
- ▶ Gesamte positive Fluss **in** einen Knoten = gesamte positive Fluss **aus** dem Knoten
(wie im Kirchhoffschen Knotenregel für Strom)

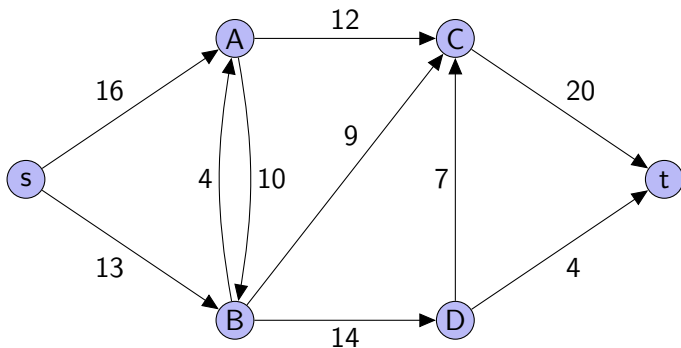
Wert eines Flusses

Definition (Wert eines Flusses)

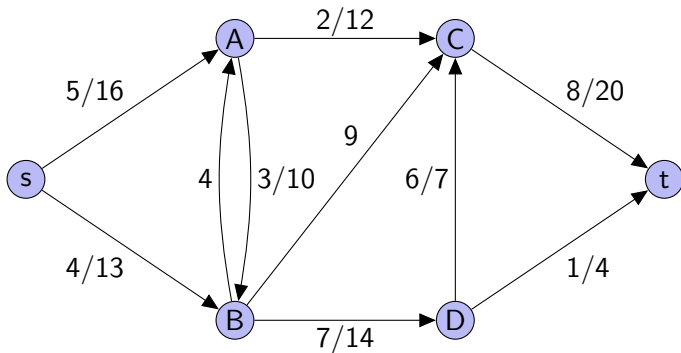
Der Wert $|f|$ des Flusses f ist der Gesamtfluss aus der Quelle s :

$$|f| = \sum_{v \in V} f(s, v).$$

Darstellung von Flüssen

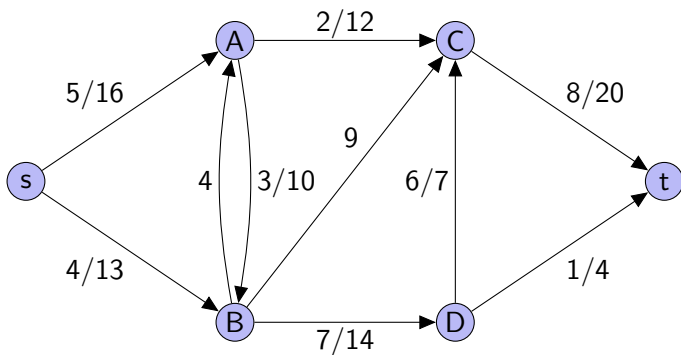


Darstellung von Flüssen



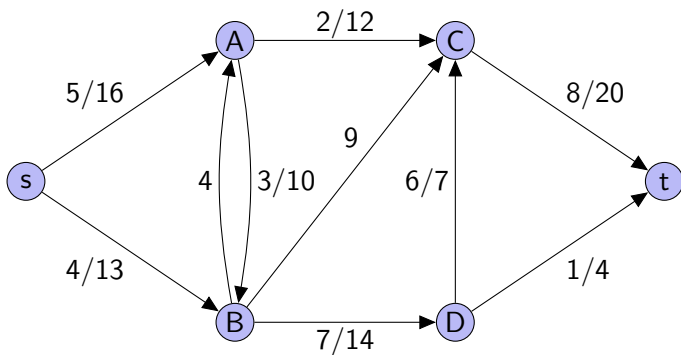
- Wir beschriften Kanten mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.

Darstellung von Flüssen



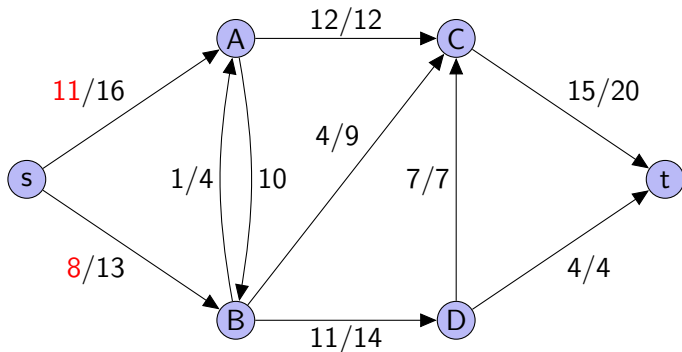
- ▶ Wir beschriften Kanten mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.
- ▶ Negative Flüsse $f(u, v) < 0$ werden nicht explizit angegeben.

Darstellung von Flüssen



- Wir beschriften Kanten mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.
- Negative Flüsse $f(u, v) < 0$ werden nicht explizit angegeben.
- Der eingezeichnete Fluss f hat den Wert $|f| = 5+4 = 9$.

Darstellung von Flüssen



- Wir beschriften Kanten mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.
- Negative Flüsse $f(u, v) < 0$ werden nicht explizit angegeben.
- Der eingezeichnete Fluss f hat den Wert $|f| = 5 + 4 = 9$.
- Der alternative Fluss f' hat den Wert $|f'| = 11 + 8 = 19$.

Maximale Flüsse

Ein **maximaler** Fluss ist einen Fluss mit maximalem Wert.

Maximale Flüsse

Ein **maximaler** Fluss ist einen Fluss mit maximalem Wert.

Problem (Maximaler Fluss)

*Finde einen **maximalen Fluss** in einem gegebenen Flussnetzwerk G .*

Maximale Flüsse

Ein **maximaler** Fluss ist einen Fluss mit maximalem Wert.

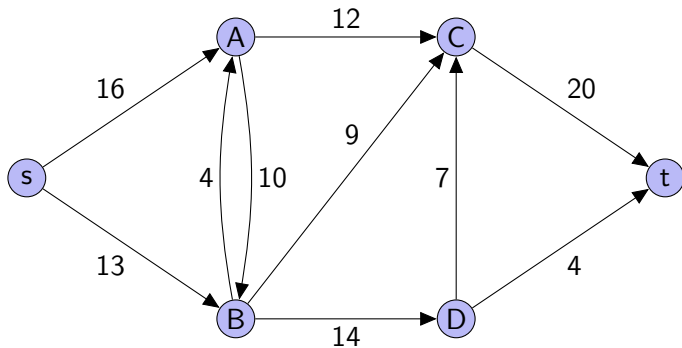
Problem (Maximaler Fluss)

Finde einen **maximalen Fluss** in einem gegebenen Flussnetzwerk G .

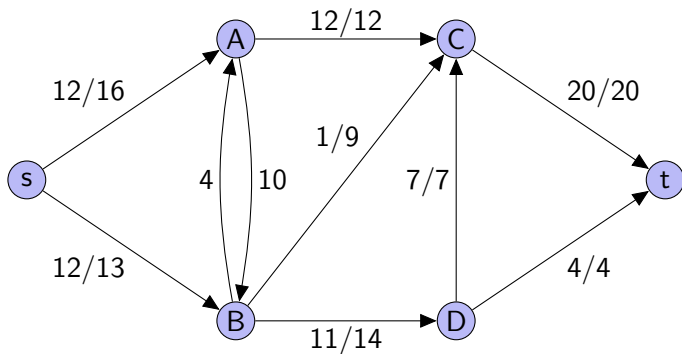
Beispiel (Anwendungen)

- ▶ Wie groß ist der maximale Datendurchsatz zwischen zwei Computern in einem Netzwerk?
- ▶ Wie kann der Verkehr in einem Straßennetz so geleitet werden, dass möglichst viele Autos in einer gegebenen Zeitspanne ein Ziel erreichen?
- ▶ Wie viele Leitungen müssen zerstört sein, damit zwei Computer nicht mehr miteinander kommunizieren können?
- ▶ Wie stark sind zwei Gruppen von Facebook-Nutzer miteinander vernetzt?

Ein maximaler Fluss

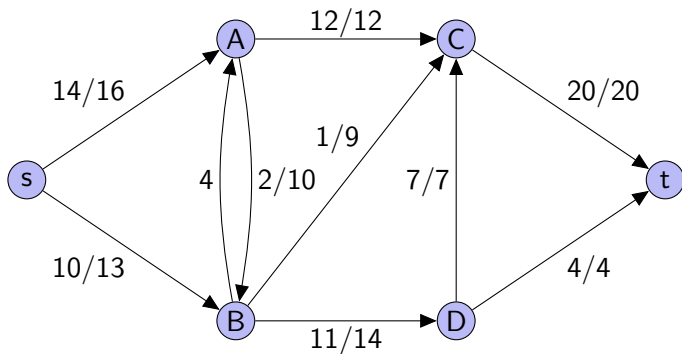


Ein maximaler Fluss



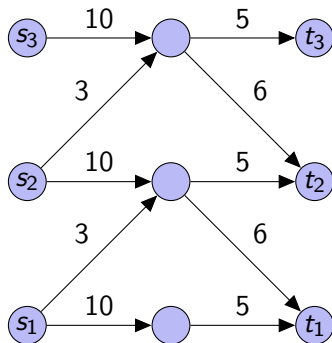
- Ein maximaler Fluss in diesem Beispiel hat den Wert $|f| = 24$.

Ein maximaler Fluss



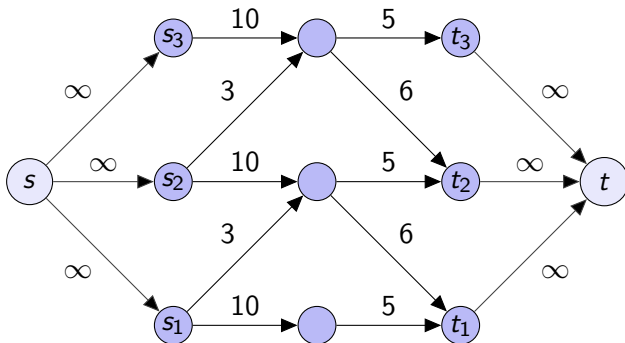
- ▶ Ein maximaler Fluss in diesem Beispiel hat den Wert $|f| = 24$.
- ▶ Es kann mehrere maximale Flüsse geben.

Mehrere Quellen und Senken



- Es kann auch Flussnetzwerke mit **mehrere** Quellen oder Senken geben.

Mehrere Quellen und Senken



- ▶ Es kann auch Flussnetzwerke mit **mehrere** Quellen oder Senken geben.
- ▶ Sie können durch eine neue „Superquelle“ und „Supersenke“ in ein übliches Flussnetzwerk überführt werden.

Flüsse zwischen Knotenmengen

Notationen

$$f(x, Y) = \sum_{y \in Y} f(x, y) \quad \text{für } Y \subseteq V$$

$$f(X, y) = \sum_{x \in X} f(x, y) \quad \text{für } X \subseteq V$$

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y) \quad \text{für } X, Y \subseteq V$$

Flüsse zwischen Knotenmengen

Notationen

$$f(x, Y) = \sum_{y \in Y} f(x, y) \quad \text{für } Y \subseteq V$$

$$f(X, y) = \sum_{x \in X} f(x, y) \quad \text{für } X \subseteq V$$

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y) \quad \text{für } X, Y \subseteq V$$

Eigenschaften von Flüssen zwischen Mengen

Für jeden Fluss f des Flussnetzwerkes $G = (V, E, c)$ gilt:

1. $f(X, X) = 0$ für $X \subseteq V$
2. $f(X, Y) = -f(Y, X)$ für $X, Y \subseteq V$
3. $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ für $X, Y, Z \subseteq V : X \cap Y = \emptyset$
4. $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$ für $X, Y, Z \subseteq V : X \cap Y = \emptyset$

Beweis: $f(X, X) = 0$

Beweis: $f(X, X) = 0$

$$f(X, X) = \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2)$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \end{aligned}$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2, x_1) \right) \end{aligned}$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2, x_1) \right) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) + f(x_2, x_1)) \end{aligned}$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2, x_1) \right) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) + f(x_2, x_1)) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) - f(x_1, x_2)) \end{aligned}$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2, x_1) \right) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) + f(x_2, x_1)) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) - f(x_1, x_2)) \\ &= 0. \end{aligned}$$

Beweis: $f(X, X) = 0$

$$\begin{aligned} f(X, X) &= \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) \right) \\ &= \frac{1}{2} \left(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1, x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2, x_1) \right) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) + f(x_2, x_1)) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} (f(x_1, x_2) - f(x_1, x_2)) \\ &= 0. \end{aligned}$$

Für den Beweis benötigen wir lediglich die Eigenschaft der Asymmetrie.

Eingehender Fluss in der Senke

Wie groß ist der an der Senke eingehende Fluss?

Eingehender Fluss in der Senke

Wie groß ist der an der Senke eingehende Fluss?

Aufgrund der Flusserhaltung in alle Zwischenknoten ist zu erwarten, dass er dem austretenden Fluss an der Quelle entspricht:

$$|f| = f(s, V) = f(V, t)$$

Eingehender Fluss in der Senke

Wie groß ist der an der Senke eingehende Fluss?

Aufgrund der Flusserhaltung in alle Zwischenknoten ist zu erwarten, dass er dem austretenden Fluss an der Quelle entspricht:

$$|f| = f(s, V) = f(V, t)$$

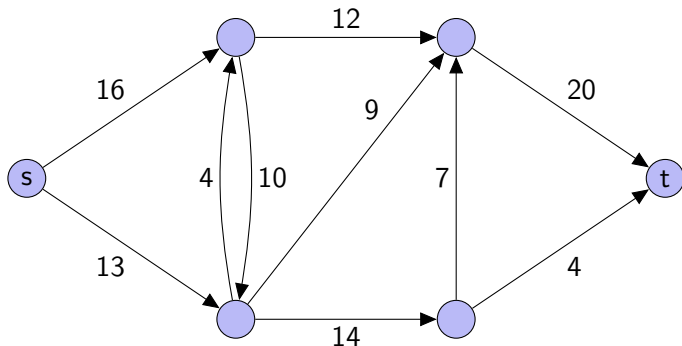
Beweis:

$f(s, V) = f(V, V) - f(V - \{s\}, V)$	Eigenschaft 3
$= -f(V - \{s\}, V)$	Eigenschaft 1
$= f(V, V - \{s\})$	Eigenschaft 2
$= f(V, t) + f(V, V - \{s, t\})$	Eigenschaft 4
$= f(V, t).$	Flusserhaltung

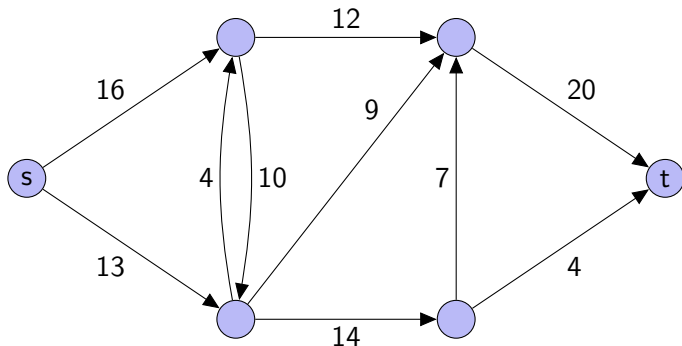
Übersicht

- 1 Flussnetzwerke
- 2 **Ford-Fulkerson-Methode**
 - Restnetzwerke
 - Algorithmus
 - Schnitte
- 3 Anwendungen und Erweiterungen
 - Edmonds-Karp-Algorithmus
 - Alternative Algorithmen

Ford-Fulkerson-Methode – Idee (1962)

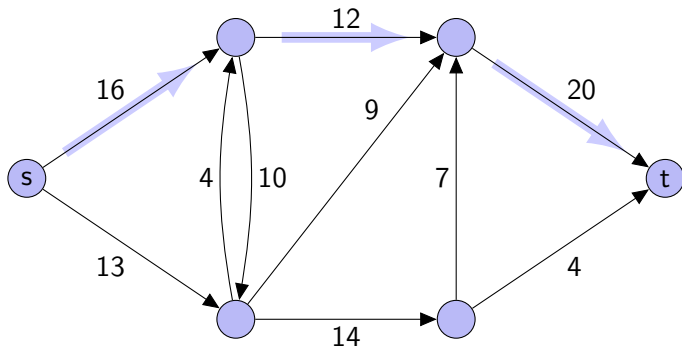


Ford-Fulkerson-Methode – Idee (1962)



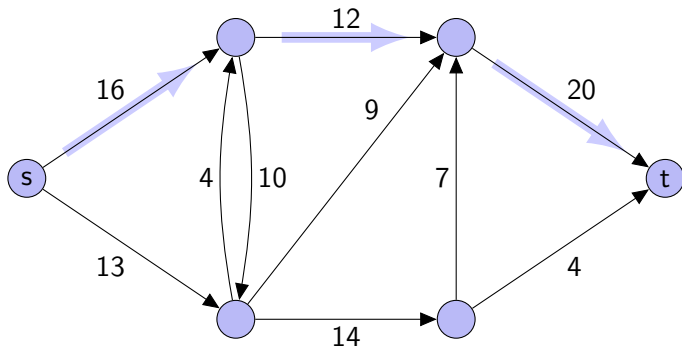
1. Suche einen Pfad p von s nach t .

Ford-Fulkerson-Methode – Idee (1962)



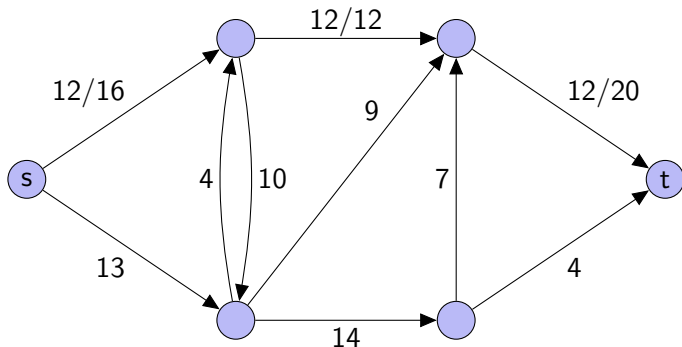
1. Suche einen Pfad p von s nach t .

Ford-Fulkerson-Methode – Idee (1962)



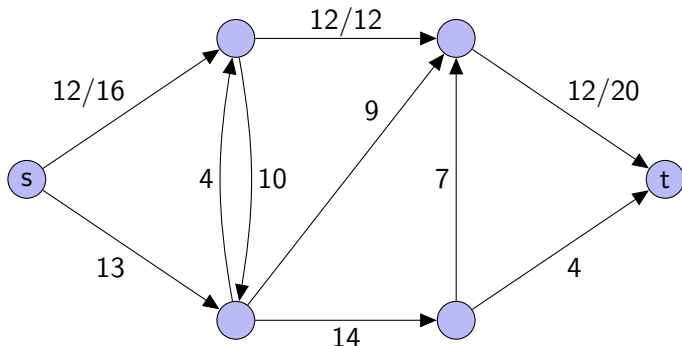
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .

Ford-Fulkerson-Methode – Idee (1962)



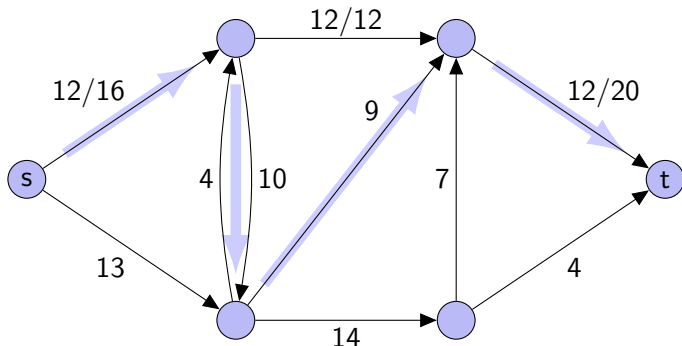
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .

Ford-Fulkerson-Methode – Idee (1962)



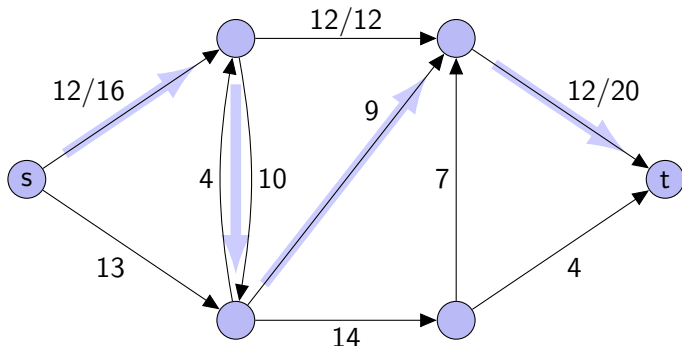
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.

Ford-Fulkerson-Methode – Idee (1962)



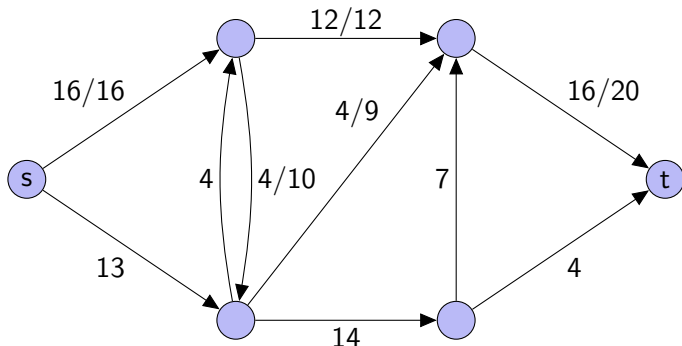
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.

Ford-Fulkerson-Methode – Idee (1962)



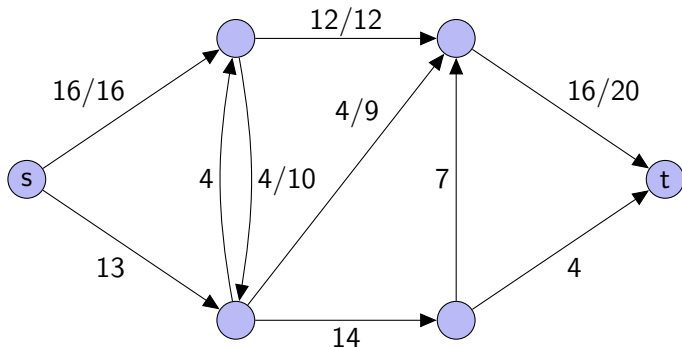
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.
4. Ergänze den Fluss der Kanten in p' um die kleinste Restkapazität in p .

Ford-Fulkerson-Methode – Idee (1962)



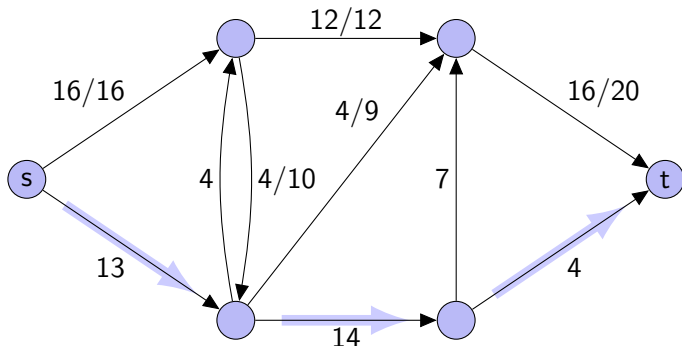
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.
4. Ergänze den Fluss der Kanten in p' um die kleinste Restkapazität in p .

Ford-Fulkerson-Methode – Idee (1962)



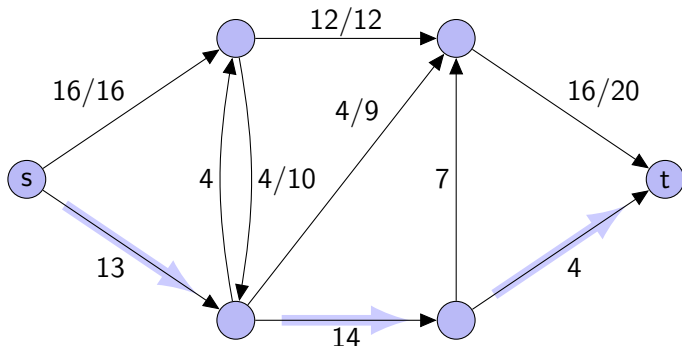
1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.
4. Ergänze den Fluss der Kanten in p' um die kleinste Restkapazität in p .
5. Wiederhole 3. und 4. bis es keinen Pfad p mehr gibt.

Ford-Fulkerson-Methode – Idee (1962)



1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.
4. Ergänze den Fluss der Kanten in p' um die kleinste Restkapazität in p .
5. Wiederhole 3. und 4. bis es keinen Pfad p mehr gibt.

Ford-Fulkerson-Methode – Idee (1962)



1. Suche einen Pfad p von s nach t .
2. Setze den Fluss der Kanten in p um die kleinste Kapazität in p .
3. Suche einen Pfad p' von s nach t , aus Kanten mit freier Kapazität.
4. Ergänze den Fluss der Kanten in p' um die kleinste Restkapazität in p .
5. Wiederhole 3. und 4. bis es keinen Pfad p mehr gibt.

Restnetzwerke

„Netzwerk minus Fluss = Restnetzwerk“

Restnetzwerke

„Netzwerk minus Fluss = Restnetzwerk“

Definition (Restnetzwerk G_f)

Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

$$c_f(u, v) = c(u, v) - f(u, v),$$

und

$$E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \},$$

Restnetzwerke

„Netzwerk minus Fluss = Restnetzwerk“

Definition (Restnetzwerk G_f)

Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

$$c_f(u, v) = c(u, v) - f(u, v),$$

und

$$E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \},$$

- ▶ $c_f(u, v)$ ist die **Restkapazität** von (u, v) in G zu Fluss f .
- ▶ E_f sind die Kanten die noch mehr Fluss aufnehmen können.

Restnetzwerk: Beispiel

Definition (Restnetzwerk G_f)

Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

- ▶ $c_f(u, v) = c(u, v) - f(u, v)$
- ▶ $E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \}.$

$c_f(u, v)$ ist die **Restkapazität** von (u, v) in G zu Fluss f .

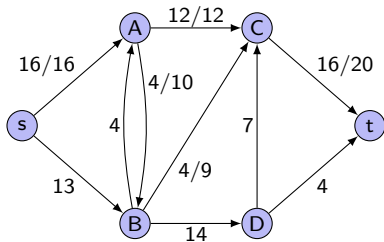
Restnetzwerk: Beispiel

Definition (Restnetzwerk G_f)

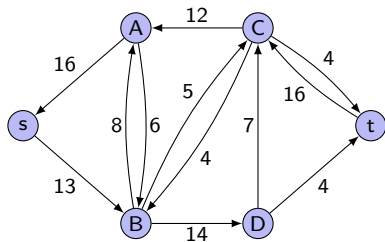
Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

- ▶ $c_f(u, v) = c(u, v) - f(u, v)$
- ▶ $E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \}$.

$c_f(u, v)$ ist die **Restkapazität** von (u, v) in G zu Fluss f .



Flussnetzwerk G



Restnetzwerk G_f

Restnetzwerk

Definition (Restnetzwerk G_f)

Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

- ▶ $c_f(u, v) = c(u, v) - f(u, v)$
- ▶ $E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \}.$

$c_f(u, v)$ ist die **Restkapazität** von (u, v) in G zu Fluss f .

Restnetzwerk

Definition (Restnetzwerk G_f)

Sei Flussnetzwerk $G = (V, E, c)$ und Fluss f . Dann ist $G_f = (V, E_f, c_f)$ das **Restnetzwerk** (auch: Residualnetzwerk) zu G und f mit:

- ▶ $c_f(u, v) = c(u, v) - f(u, v)$
- ▶ $E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \}.$

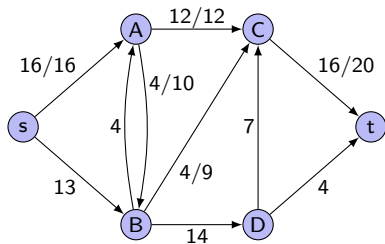
$c_f(u, v)$ ist die **Restkapazität** von (u, v) in G zu Fluss f .

Kanten im Restnetzwerk

- ▶ Falls $f(u, v) < c(u, v)$, dann folgt $c_f(u, v) > 0$ und $(u, v) \in E_f$
- ▶ Falls $f(u, v) > 0$, dann $f(v, u) < 0$, und damit $c_f(v, u) > 0$ und $(v, u) \in E_f$
- ▶ Falls weder $(u, v) \notin E$ noch $(v, u) \notin E$, dann $c_f(u, v) = c_f(v, u) = 0$

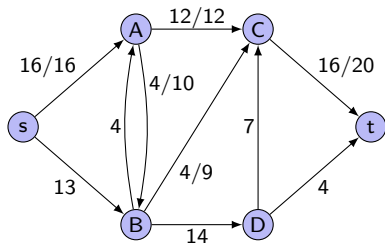
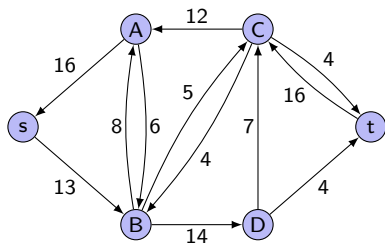
Also $|E_f| \leq 2 \cdot |E|$.

Augmentierende Pfade

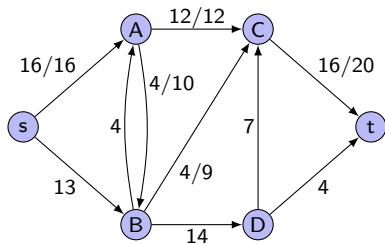
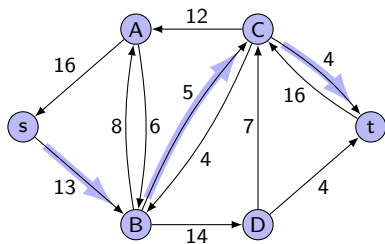


Flussnetzwerk G

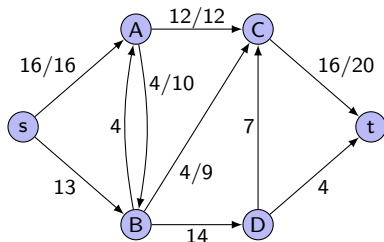
Augmentierende Pfade

Flussnetzwerk G Restnetzwerk G_f

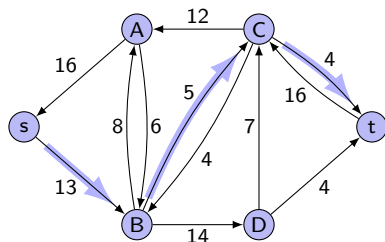
Augmentierende Pfade

Flussnetzwerk G Restnetzwerk G_f

Augmentierende Pfade



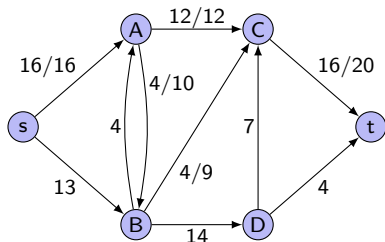
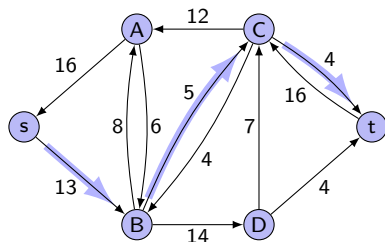
Flussnetzwerk G



Restnetzwerk G_f

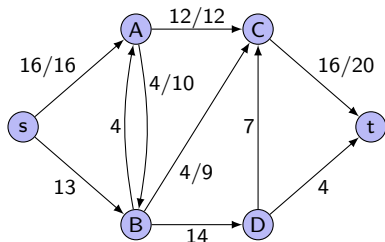
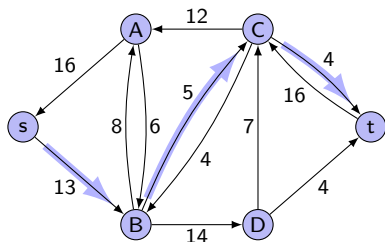
- Ein s - t -Pfad p in Restnetzwerk G_f heißt **augmentierender Pfad** (vergrößernder Pfad).

Augmentierende Pfade

Flussnetzwerk G Restnetzwerk G_f

- Ein s - t -Pfad p in Restnetzwerk G_f heißt **augmentierender Pfad** (vergrößernder Pfad).
- $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$ heißt **Restkapazität von p** .

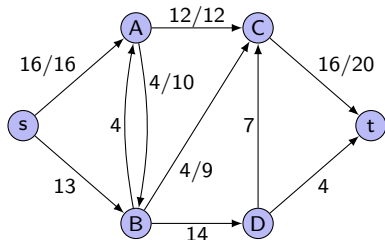
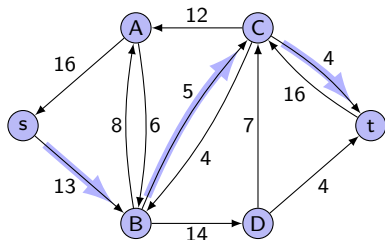
Augmentierende Pfade

Flussnetzwerk G Restnetzwerk G_f

- Ein s - t -Pfad p in Restnetzwerk G_f heißt **augmentierender Pfad** (vergrößernder Pfad).
- $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$ heißt **Restkapazität von p** .

Der Pfad im obigen Beispiel hat die Restkapazität 4.

Augmentierende Pfade

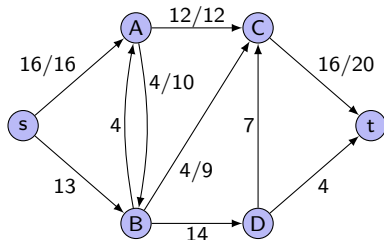
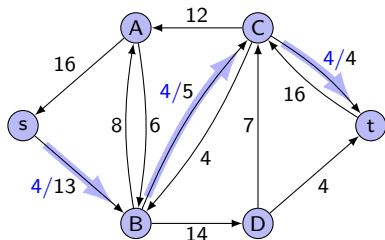
Flussnetzwerk G Restnetzwerk G_f

Sei G ein Flussnetzwerk, f ein Fluss in G , p ein augmentierender Pfad in G_f . Sei:

$$f_p(u, v) = \begin{cases} c_f(p) & \text{falls } (u, v) \text{ auf } p \\ -c_f(p) & \text{falls } (v, u) \text{ auf } p \\ 0 & \text{sonst} \end{cases}$$

Dann ist f_p ein **Fluss in Restnetzwerk** G_f mit dem Wert $|f_p| = c_f(p) > 0$.

Augmentierende Pfade

Flussnetzwerk G Restnetzwerk G_f

Sei G ein Flussnetzwerk, f ein Fluss in G , p ein augmentierender Pfad in G_f . Sei:

$$f_p(u, v) = \begin{cases} c_f(p) & \text{falls } (u, v) \text{ auf } p \\ -c_f(p) & \text{falls } (v, u) \text{ auf } p \\ 0 & \text{sonst} \end{cases}$$

Dann ist f_p ein **Fluss in Restnetzwerk** G_f mit dem Wert $|f_p| = c_f(p) > 0$.

Ford-Fulkerson-Theorem

Idee: ergänze ein Fluss f in G um den Fluss f' im Restnetzwerk G_f .

Ford-Fulkerson-Theorem

Idee: ergänze ein Fluss f in G um den Fluss f' im Restnetzwerk G_f .

Sei $f_1, f_2 : V \times V \rightarrow \mathbb{R}$ zwei Flüsse. Die **Flusssumme** $f_1 + f_2$ ist definiert durch: $(f_1 + f_2)(u, v) = f_1(u, v) + f_2(u, v)$.

Ford-Fulkerson-Theorem

Idee: ergänze ein Fluss f in G um den Fluss f' im Restnetzwerk G_f .

Sei $f_1, f_2 : V \times V \rightarrow \mathbb{R}$ zwei Flüsse. Die **Flusssumme** $f_1 + f_2$ ist definiert durch: $(f_1 + f_2)(u, v) = f_1(u, v) + f_2(u, v)$.

[

Ford-Fulkerson Theorem] Für Flussnetzwerk G , f ein Fluss in G und f' ein Fluss in G_f :

$f + f'$ ist ein Fluss in G (mit dem Wert $|f + f'|$).

Ford-Fulkerson-Theorem

Idee: ergänze ein Fluss f in G um den Fluss f' im Restnetzwerk G_f .

Sei $f_1, f_2 : V \times V \rightarrow \mathbb{R}$ zwei Flüsse. Die **Flusssumme** $f_1 + f_2$ ist definiert durch: $(f_1 + f_2)(u, v) = f_1(u, v) + f_2(u, v)$.

[

Ford-Fulkerson Theorem] Für Flussnetzwerk G , f ein Fluss in G und f' ein Fluss in G_f :

$f + f'$ ist ein Fluss in G (mit dem Wert $|f + f'|$).

Beweis.

Wir zeigen, dass $f + f'$ beschränkt, asymmetrisch und flusserhaltend ist (s. nächste Folie). □

1. Asymmetrie:

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\&= -f(v, u) - f'(v, u) \\&= -(f(v, u) + f'(v, u)) \\&= -(f + f')(v, u)\end{aligned}$$

1. Asymmetrie:

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &= -f(v, u) - f'(v, u) \\ &= -(f(v, u) + f'(v, u)) \\ &= -(f + f')(v, u)\end{aligned}$$

2. Flusserhaltung:

$$(f + f')(u, V) = f(u, V) + f'(u, V) = 0 \quad | \quad \forall u \in V - \{s, t\}$$

1. Asymmetrie:

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &= -f(v, u) - f'(v, u) \\ &= -(f(v, u) + f'(v, u)) \\ &= -(f + f')(v, u)\end{aligned}$$

2. Flusserhaltung:

$$(f + f')(u, V) = f(u, V) + f'(u, V) = 0 \quad | \quad \forall u \in V - \{s, t\}$$

3. Beschränkung:

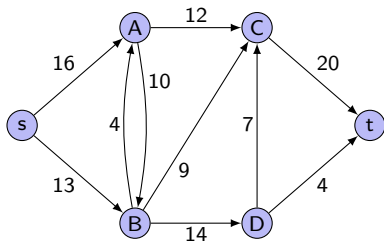
$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &\leq f(u, v) + c_f(u, v) \\ &= f(u, v) + (c(u, v) - f(u, v)) \\ &= c(u, v).\end{aligned}$$

Die Ford-Fulkerson-Methode

Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



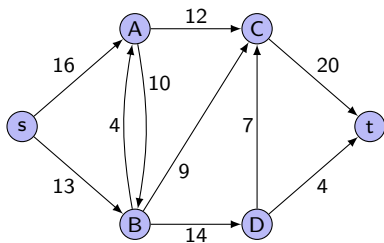
Flussnetzwerk G

Die Ford-Fulkerson-Methode

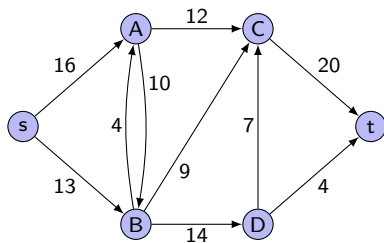
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



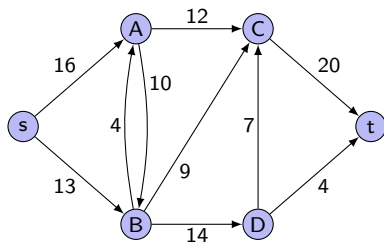
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

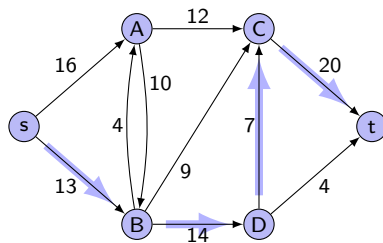
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



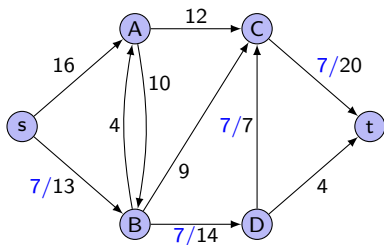
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

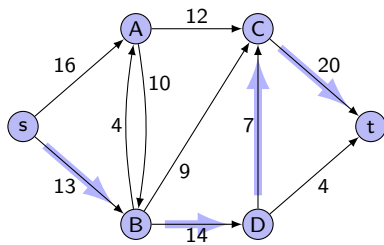
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



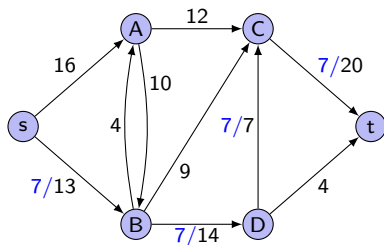
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

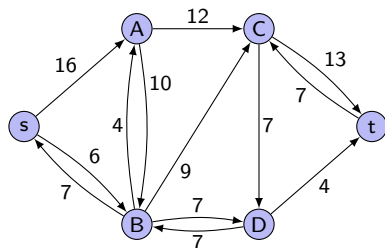
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



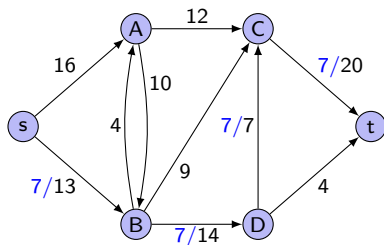
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

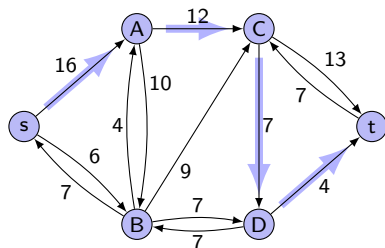
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



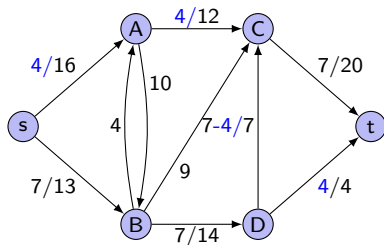
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

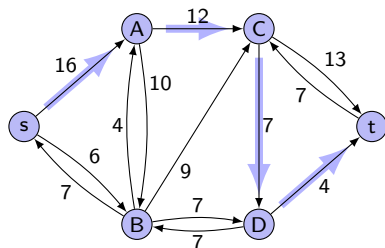
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



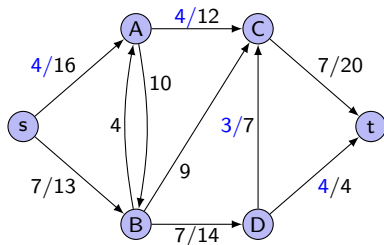
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

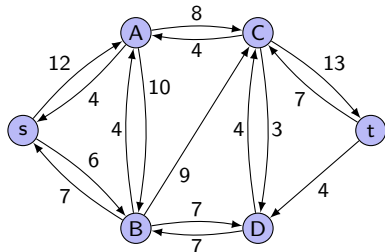
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



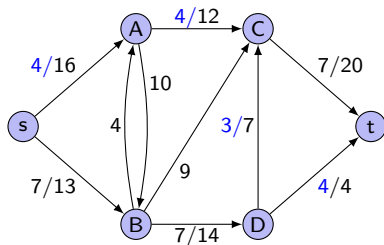
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

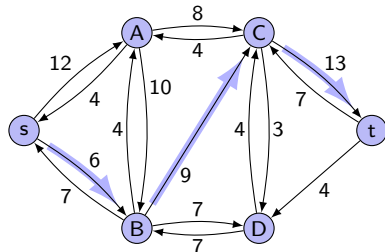
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



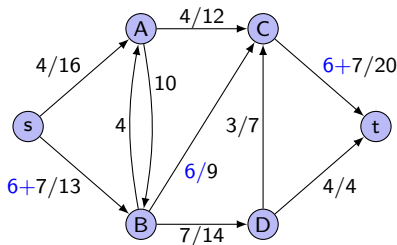
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

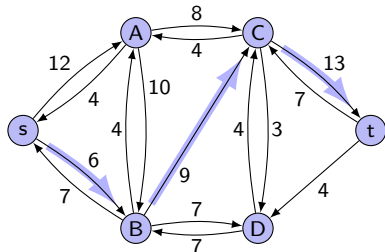
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



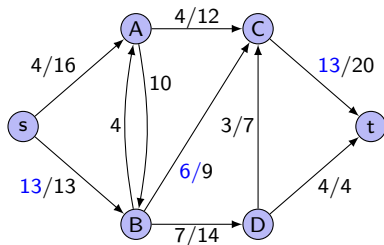
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

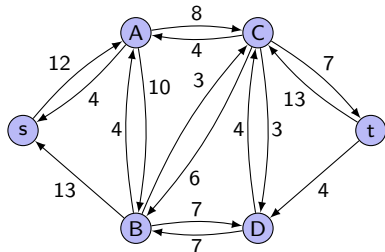
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



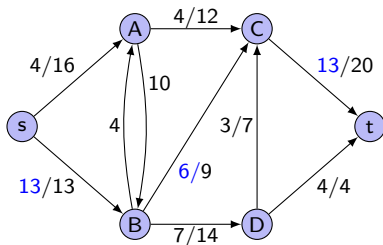
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

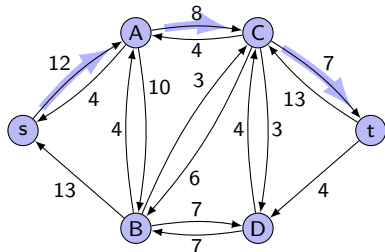
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



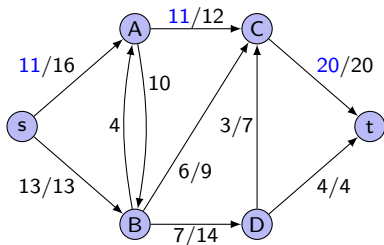
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

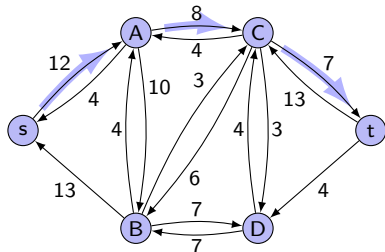
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



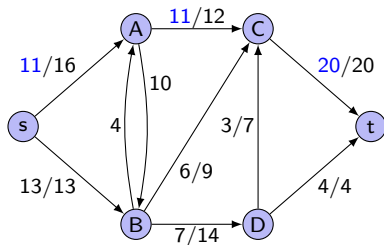
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

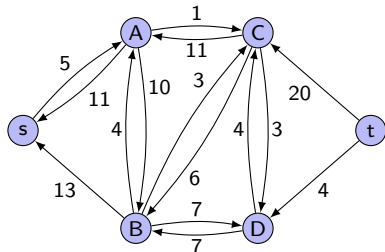
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



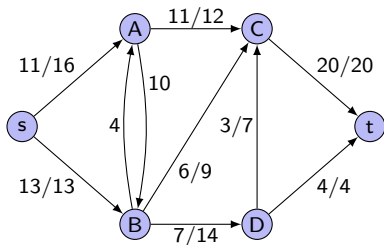
Restnetzwerk G_f

Die Ford-Fulkerson-Methode

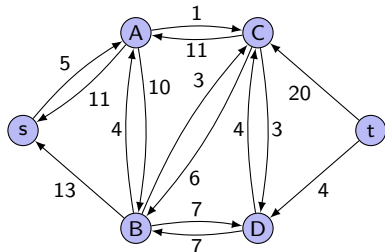
Algorithmus

Initialisiere Fluss f zu 0

while es gibt einen augmentierenden Pfad p
 do augmentiere f entlang p // $f := f + f_p$
return f



Flussnetzwerk G



Restnetzwerk G_f

Implementierung Ford-Fulkerson-Methode

```
1 int[n,n] maxFlow(List adjLst[n], int n, int s, int t) {
2     int flow[n,n] = 0, path[];
3     int cfp; // Restkapazität des Pfades
4
5     while (true) {
6         // Finde augmentierenden Pfad und dessen Restkapazität
7         (path, cfp) = augmentPfad(adjLst, flow, s, t);
8         if (cfp == 0) { // kein Pfad gefunden
9             return flow;
10        }
11
12        // addiere Restkapazität entlang des Pfades zum Fluss
13        for (int i = 1; i < path.length; i++) {
14            int u = path[i-1], v = path[i];
15            flow[u,v] = flow[u,v] + cfp;
16            flow[v,u] = -flow[u,v];
17        }
18    }
19 }
```

Laufzeit der Ford-Fulkerson-Methode

Ein Flussproblem ist **integral**, wenn alle Kapazitäten ganzzahlig sind.

Laufzeit der Ford-Fulkerson-Methode

Ein Flussproblem ist **integral**, wenn alle Kapazitäten ganzzahlig sind.

Zeitkomplexität

Sei f^* der durch die Ford-Fulkerson-Methode bestimmte Fluss zu einem integralen Flussproblem, so benötigt die Methode höchstens $|f^*|$ Iterationen und es ergibt sich eine worst-case Laufzeit von $O(|E| \cdot |f^*|)$.

Laufzeit der Ford-Fulkerson-Methode

Ein Flussproblem ist **integral**, wenn alle Kapazitäten ganzzahlig sind.

Zeitkomplexität

Sei f^* der durch die Ford-Fulkerson-Methode bestimmte Fluss zu einem integralen Flussproblem, so benötigt die Methode höchstens $|f^*|$ Iterationen und es ergibt sich eine worst-case Laufzeit von $O(|E| \cdot |f^*|)$.

Beweis.

In jeder Iteration wird der Wert des Flusses um $c_f(p) \geq 1$ erhöht. Er ist anfangs 0 und am Ende f^* . Für jede Iteration, können augmentierende Pfade in $O(|E|)$ bestimmt werden. □

Laufzeit der Ford-Fulkerson-Methode

Ein Flussproblem ist **integral**, wenn alle Kapazitäten ganzzahlig sind.

Zeitkomplexität

Sei f^* der durch die Ford-Fulkerson-Methode bestimmte Fluss zu einem integralen Flussproblem, so benötigt die Methode höchstens $|f^*|$ Iterationen und es ergibt sich eine worst-case Laufzeit von $O(|E| \cdot |f^*|)$.

Beweis.

In jeder Iteration wird der Wert des Flusses um $c_f(p) \geq 1$ erhöht. Er ist anfangs 0 und am Ende f^* . Für jede Iteration, können augmentierende Pfade in $O(|E|)$ bestimmt werden. □

Korollar

Bei **rationalen** Kapazitäten terminiert die Ford-Fulkerson-Methode. Brüche können durch Multiplikation aufgehoben werden.

Laufzeit der Ford-Fulkerson-Methode

Ein Flussproblem ist **integral**, wenn alle Kapazitäten ganzzahlig sind.

Zeitkomplexität

Sei f^* der durch die Ford-Fulkerson-Methode bestimmte Fluss zu einem integralen Flussproblem, so benötigt die Methode höchstens $|f^*|$ Iterationen und es ergibt sich eine worst-case Laufzeit von $O(|E| \cdot |f^*|)$.

Beweis.

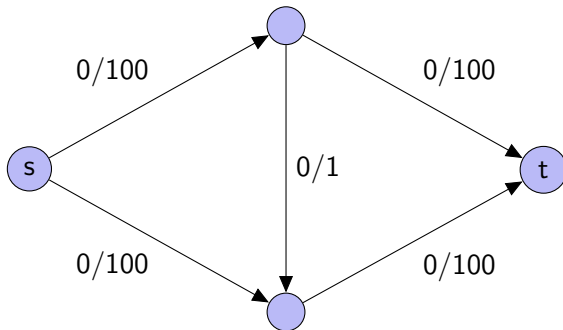
In jeder Iteration wird der Wert des Flusses um $c_f(p) \geq 1$ erhöht. Er ist anfangs 0 und am Ende f^* . Für jede Iteration, können augmentierende Pfade in $O(|E|)$ bestimmt werden. □

Korollar

Bei **rationalen** Kapazitäten terminiert die Ford-Fulkerson-Methode. Brüche können durch Multiplikation aufgehoben werden.

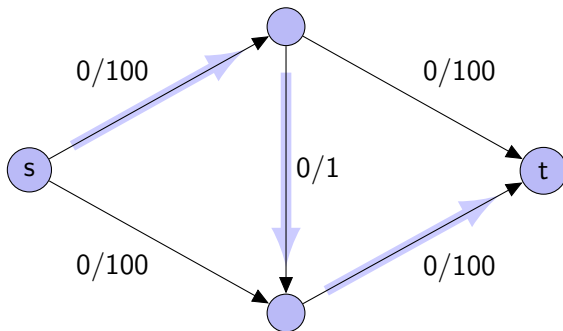
- Für ein **integrales** Flussproblem bestimmt die Ford-Fulkerson-Methode einen Fluss f , sodass jedes $f(u, v)$ **ganzzahlig** ist.

Laufzeit der Ford-Fulkerson-Methode



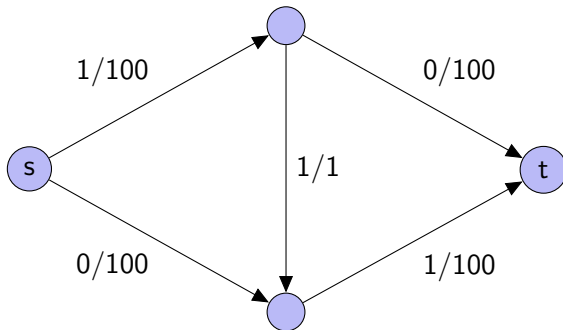
Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



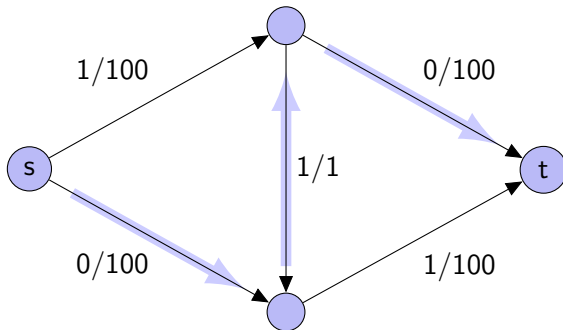
Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



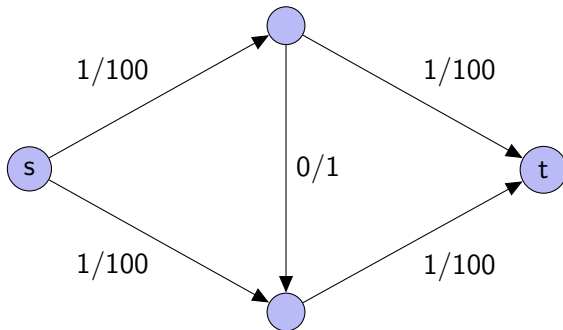
Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



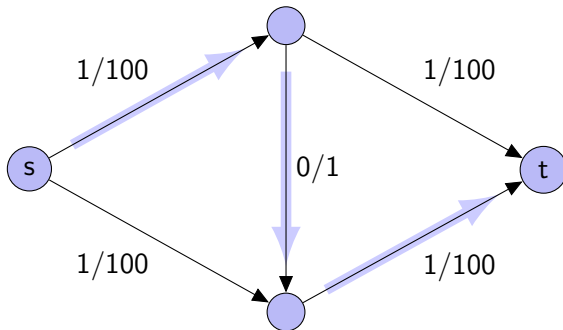
Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



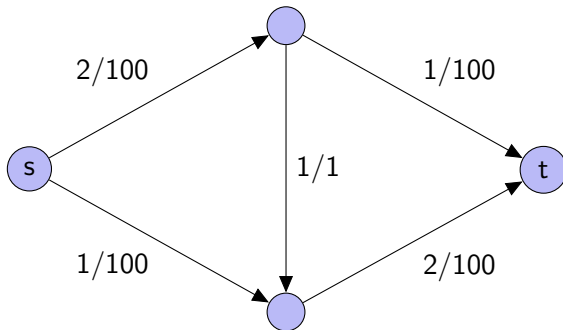
Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Laufzeit der Ford-Fulkerson-Methode



Die Worst-Case-Laufzeit ist abhängig vom Wert eines maximalen Flusses, da der Wert des Flusses im schlimmsten Fall sich jeweils nur um eine Einheit erhöht.

Korrektheit Ford-Fulkerson Methode

Die Ford-Fulkerson Methode erweitert sukzessive den Fluss in G um augmentierende Pfade im Restnetzwerk G_f bis es keine solche Pfade mehr gibt.

Korrektheit Ford-Fulkerson Methode

Die Ford-Fulkerson Methode erweitert sukzessive den Fluss in G um augmentierende Pfade im Restnetzwerk G_f bis es keine solche Pfade mehr gibt.

Ist das korrekt?

Korrektheit Ford-Fulkerson Methode

Die Ford-Fulkerson Methode erweitert sukzessive den Fluss in G um augmentierende Pfade im Restnetzwerk G_f bis es keine solche Pfade mehr gibt.

Ist das korrekt?

Wir werden zeigen, dass ein Fluss in G genau dann **maximal** ist, wenn sein Restnetzwerk **keine augmentierende Pfade** enthält.

Korrektheit Ford-Fulkerson Methode

Die Ford-Fulkerson Methode erweitert sukzessive den Fluss in G um augmentierende Pfade im Restnetzwerk G_f bis es keine solche Pfade mehr gibt.

Ist das korrekt?

Wir werden zeigen, dass ein Fluss in G genau dann **maximal** ist, wenn sein Restnetzwerk **keine augmentierende Pfade** enthält.

Dazu benutzen wir **Schnitte**.

Schnitte in Flussnetzwerken

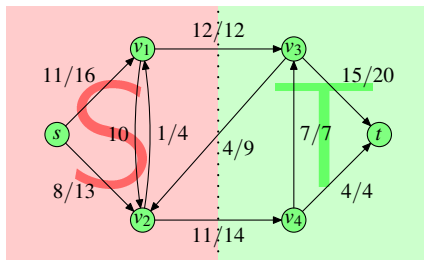
Definition

Ein **Schnitt** (S, T) in einem Flussnetzwerk $G = (V, E, c)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

Schnitte in Flussnetzwerken

Definition

Ein **Schnitt** (S, T) in einem Flussnetzwerk $G = (V, E, c)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

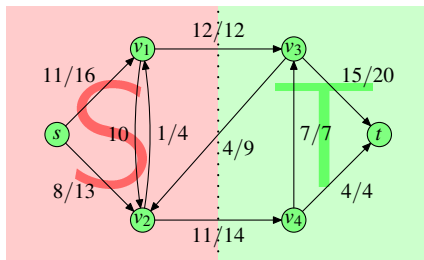


Schnitte in Flussnetzwerken

Definition

Ein **Schnitt** (S, T) in einem Flussnetzwerk $G = (V, E, c)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

- Wenn f ein Fluss in G ist, dann ist $f(S, T)$ der **Fluss über** (S, T) .

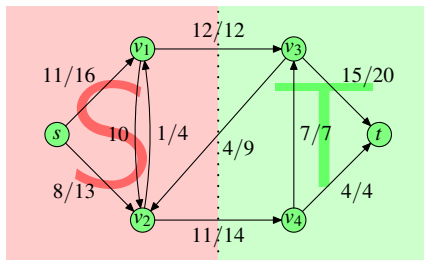


Schnitte in Flussnetzwerken

Definition

Ein **Schnitt** (S, T) in einem Flussnetzwerk $G = (V, E, c)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

- ▶ Wenn f ein Fluss in G ist, dann ist $f(S, T)$ der **Fluss über (S, T)** .
- ▶ Die **Kapazität von (S, T)** ist $c(S, T)$.

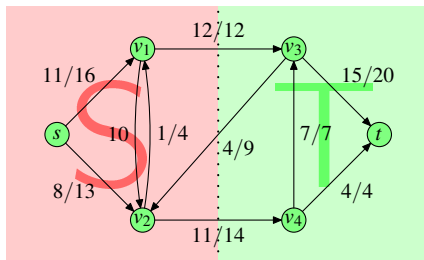


Schnitte in Flussnetzwerken

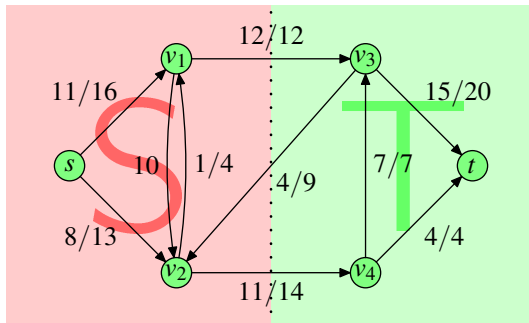
Definition

Ein **Schnitt** (S, T) in einem Flussnetzwerk $G = (V, E, c)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

- ▶ Wenn f ein Fluss in G ist, dann ist $f(S, T)$ der **Fluss über (S, T)** .
- ▶ Die **Kapazität von (S, T)** ist $c(S, T)$.
- ▶ Ein **minimaler Schnitt** ist ein Schnitt mit minimaler Kapazität.

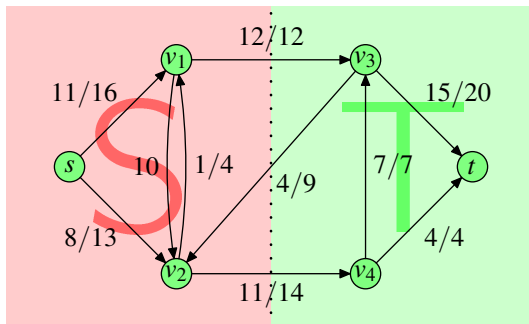


Schnitte in Flussnetzwerken



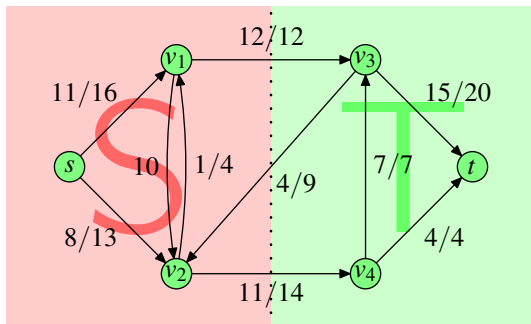
S	:	$\{s, v_1, v_2\}$	
T	:	$\{t, v_3, v_4\}$	
Fluss	:		
Kapazität	:		

Schnitte in Flussnetzwerken



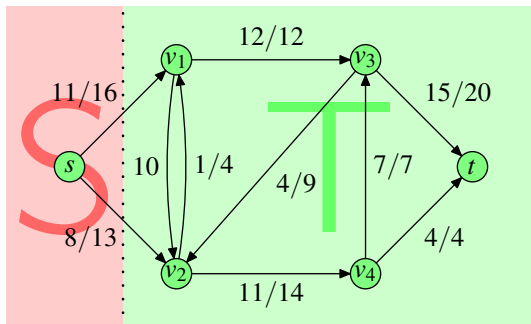
S	:	$\{s, v_1, v_2\}$	
T	:	$\{t, v_3, v_4\}$	
Fluss	:	$12+11-4$	
Kapazität	:		

Schnitte in Flussnetzwerken



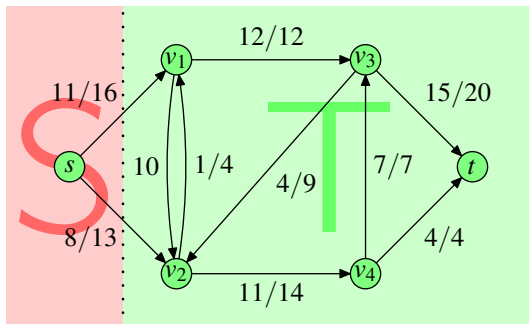
S	:	$\{s, v_1, v_2\}$	
T	:	$\{t, v_3, v_4\}$	
Fluss	:	19	
Kapazität	:	$12+14$	

Schnitte in Flussnetzwerken



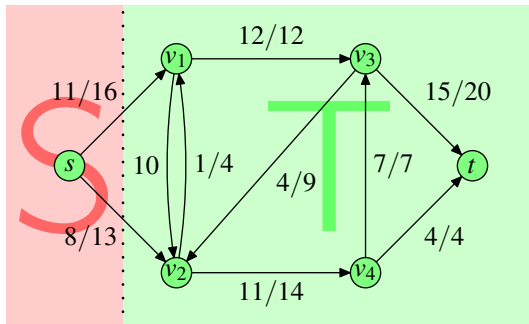
S	:	$\{s, v_1, v_2\}$	$\{s\}$	
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	
Fluss	:	19		
Kapazität	:	26		

Schnitte in Flussnetzwerken



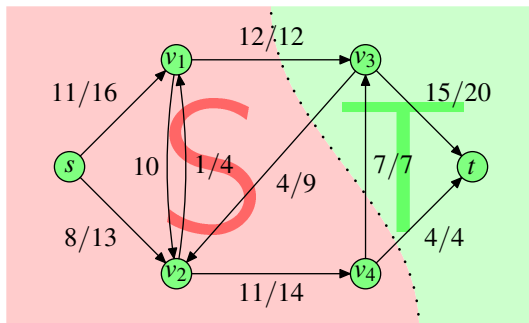
S	:	$\{s, v_1, v_2\}$	$\{s\}$	
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	
Fluss	:	19	11+8	
Kapazität	:	26		

Schnitte in Flussnetzwerken



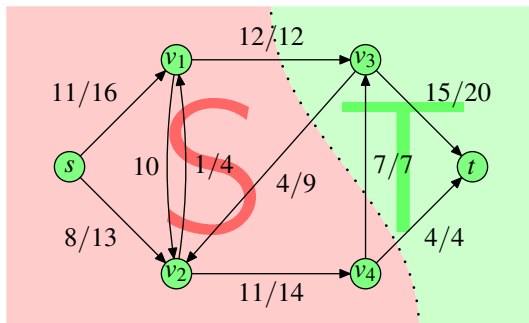
S	:	$\{s, v_1, v_2\}$	$\{s\}$	
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	
Fluss	:	19	19	
Kapazität	:	26	13+16	

Schnitte in Flussnetzwerken



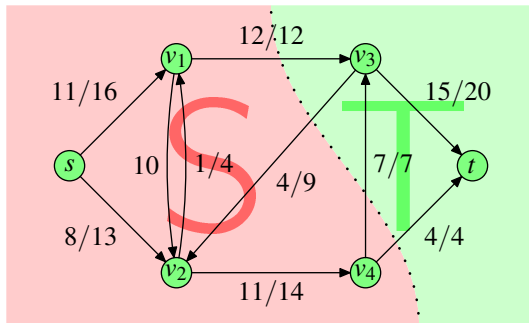
S	:	$\{s, v_1, v_2\}$	$\{s\}$	$\{s, v_1, v_2, v_4\}$
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	$\{t, v_3\}$
Fluss	:	19	19	
Kapazität	:	26	29	

Schnitte in Flussnetzwerken



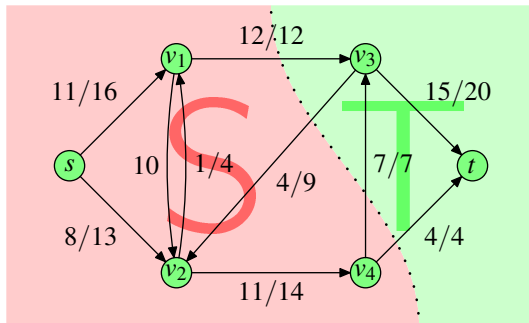
S	:	$\{s, v_1, v_2\}$	$\{s\}$	$\{s, v_1, v_2, v_4\}$
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	$\{t, v_3\}$
Fluss	:	19	19	$12-4+7+4$
Kapazität	:	26	29	

Schnitte in Flussnetzwerken



S	:	$\{s, v_1, v_2\}$	$\{s\}$	$\{s, v_1, v_2, v_4\}$
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	$\{t, v_3\}$
Fluss	:	19	19	19
Kapazität	:	26	29	$12+7+4$

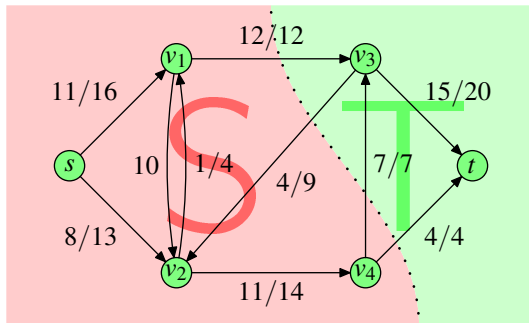
Schnitte in Flussnetzwerken



S	:	$\{s, v_1, v_2\}$	$\{s\}$	$\{s, v_1, v_2, v_4\}$
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	$\{t, v_3\}$
Fluss	:	19	19	19
Kapazität	:	26	29	23

► Für den Fluss über einen Schnitt gilt: $f(S, T) = |f|$

Schnitte in Flussnetzwerken



S	:	$\{s, v_1, v_2\}$	$\{s\}$	$\{s, v_1, v_2, v_4\}$
T	:	$\{t, v_3, v_4\}$	$\{t, v_1, v_2, v_3, v_4\}$	$\{t, v_3\}$
Fluss	:	19	19	19
Kapazität	:	26	29	23

- Für den Fluss über einen Schnitt gilt: $f(S, T) = |f| \leq c(S, T)$.

Max-flow Min-cut Theorem

Max-flow Min-cut Theorem

Sei f ein Fluss im Flussnetzwerk $G = (V, E, c)$, dann sind äquivalent:

1. f ist ein maximaler Fluss.
2. In Restnetzwerk G_f gibt es keinen augmentierenden Pfad.
3. G hat einen Schnitt (S, T) mit $|f| = c(S, T)$, d. h. (S, T) ist minimal.

Max-flow Min-cut Theorem

Max-flow Min-cut Theorem

Sei f ein Fluss im Flussnetzwerk $G = (V, E, c)$, dann sind äquivalent:

1. f ist ein maximaler Fluss.
2. In Restnetzwerk G_f gibt es keinen augmentierenden Pfad.
3. G hat einen Schnitt (S, T) mit $|f| = c(S, T)$, d. h. (S, T) ist minimal.

Folgerungen

1. Die Kapazität eines minimalen Schnittes ist gleich dem Wert eines maximalen Flusses.
2. Falls die Ford–Fulkerson–Methode terminiert, berechnet sie einen maximalen Fluss.

Max-flow Min-cut Theorem

Max-flow Min-cut Theorem

Sei f ein Fluss im Flussnetzwerk $G = (V, E, c)$, dann sind äquivalent:

1. f ist ein maximaler Fluss.
2. In Restnetzwerk G_f gibt es keinen augmentierenden Pfad.
3. G hat einen Schnitt (S, T) mit $|f| = c(S, T)$, d. h. (S, T) ist minimal.

1. $\Rightarrow$ 2. (Widerspruchsbeweis).

Sei f ein maximaler Fluss in G und p einen augmentierender Pfad in G_f .

$\Rightarrow f + f_p$ ist ein Fluss in G mit $|f + f_p| > |f|$.

$\Rightarrow$ **Widerspruch!** Denn f ist ein maximaler Fluss.



Max-flow Min-cut Theorem

Max-flow Min-cut Theorem

Sei f ein Fluss im Flussnetzwerk $G = (V, E, c)$, dann sind äquivalent:

1. f ist ein maximaler Fluss.
2. In Restnetzwerk G_f gibt es keinen augmentierenden Pfad.
3. G hat einen Schnitt (S, T) mit $|f| = c(S, T)$, d. h. (S, T) ist minimal.

2. $\Rightarrow$ 3.

Nehme an, es gibt keinen s - t -Pfad (d.h. augmentierten Pfad) in G_f .

Sei $S := \{v \in V \mid \exists s\text{-}v\text{-Pfad in } G_f\}$ und $T := V - S$, dann gilt:

1. $\forall u \in S, v \in T$ gilt: $c_f(u, v) = 0 \Rightarrow f(u, v) = c(u, v)$.
 2. (S, T) ist ein Schnitt und somit gilt $f(S, T) = |f|$.
- $\Rightarrow c(S, T) = f(S, T) = |f|$.



Max-flow Min-cut Theorem

Max-flow Min-cut Theorem

Sei f ein Fluss im Flussnetzwerk $G = (V, E, c)$, dann sind äquivalent:

1. f ist ein maximaler Fluss.
2. In Restnetzwerk G_f gibt es keinen augmentierenden Pfad.
3. G hat einen Schnitt (S, T) mit $|f| = c(S, T)$, d. h. (S, T) ist minimal.

3. $\Rightarrow$ 1.

Sei f' ein beliebiger Fluss in G dann gilt:

$$|f'| = f'(S, T) = \sum_{u \in S} \sum_{v \in T} f'(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)$$

Da $|f| = c(S, T)$ und $\forall f' : |f'| \leq c(S, T)$, folgt f ist maximal.



Übersicht

- 1 Flussnetzwerke
- 2 Ford-Fulkerson-Methode
 - Restnetzwerke
 - Algorithmus
 - Schnitte
- 3 Anwendungen und Erweiterungen
 - Edmonds-Karp-Algorithmus
 - Alternative Algorithmen

Edmonds-Karp-Algorithmus

Edmonds-Karp-Algorithmus

Eine Implementierung der Ford-Fulkerson-Methode, die zur Bestimmung augmentierender Pfade eine **Breitensuche** nutzt. Laufzeit: $O(|V| \cdot |E|^2)$.

Edmonds-Karp-Algorithmus

Edmonds-Karp-Algorithmus

Eine Implementierung der Ford-Fulkerson-Methode, die zur Bestimmung augmentierender Pfade eine **Breitensuche** nutzt. Laufzeit: $O(|V| \cdot |E|^2)$.
Sie erweitert stets den Fluss entlang kürzester Pfade.

Edmonds-Karp-Algorithmus

Edmonds-Karp-Algorithmus

Eine Implementierung der Ford-Fulkerson-Methode, die zur Bestimmung augmentierender Pfade eine **Breitensuche** nutzt. Laufzeit: $O(|V| \cdot |E|^2)$.
Sie erweitert stets den Fluss entlang kürzester Pfade.

Lemma

Im Edmonds-Karp-Algorithmus steigt für alle Knoten $v \in V - \{s, t\}$ der Abstand (d.h. Anzahl der Kanten) des kürzesten Pfades von s nach v im Restnetzwerk G_f **monoton** mit jeder Flusserweiterung.

Edmonds-Karp-Algorithmus

Edmonds-Karp-Algorithmus

Eine Implementierung der Ford-Fulkerson-Methode, die zur Bestimmung augmentierender Pfade eine **Breitensuche** nutzt. Laufzeit: $O(|V| \cdot |E|^2)$.
Sie erweitert stets den Fluss entlang kürzester Pfade.

Lemma

Im Edmonds-Karp-Algorithmus steigt für alle Knoten $v \in V - \{s, t\}$ der Abstand (d.h. Anzahl der Kanten) des kürzesten Pfades von s nach v im Restnetzwerk G_f **monoton** mit jeder Flusserweiterung.

Theorem

Die Gesamtzahl der Flusserweiterungen im Edmonds-Karp-Algorithmus für das Flussnetzwerk $G = (V, E, c)$ ist in $O(|V| \cdot |E|)$.

Andere Max-Flow Algorithmen

Flüsse und Schnitte in Netzwerken - Wikipedia - Mozilla Firefox

de.wikipedia.org/wiki/Flüsse_und_Schnitte_in_Netzwerken

Max-Flow Algorithmen nach Veröffentlichung [Bearbeiten]

Jahr	Autor(en)	Name	Laufzeiten
1956	Ford, Fulkerson	Algorithmus von Ford und Fulkerson	$\mathcal{O}(m \cdot n \cdot u_{\max})$, falls alle Kapazitäten ganzzahlig sind
1969	Edmonds, Karp	Algorithmus von Edmonds und Karp	$\mathcal{O}(m^2 \cdot n)$
1970	Dinic	Algorithmus von Dinic	$\mathcal{O}(m \cdot n^2)$
1973	Dinic, Gabow		$\mathcal{O}(m \cdot n \cdot \log(u_{\max}))$
1974	Karzanov		$\mathcal{O}(n^3)$
1977	Cherkassky		$\mathcal{O}(n^2 \cdot \sqrt{m})$
1980	Galil, Naamad		$\mathcal{O}(m \cdot n \cdot \log(n)^2)$
1983	Sleator, Tarjan		$\mathcal{O}(m \cdot n \cdot \log(n))$
1986	Goldberg, Tarjan	Goldberg-Tarjan-Algorithmus	$\mathcal{O}\left(m \cdot n \cdot \log\left(\frac{n^2}{m}\right)\right)$
1987	Ahuja, Orlin		$\mathcal{O}(m \cdot n + n^2 \cdot \log(u_{\max}))$
1987	Ahuja, Orlin, Tarjan		$\mathcal{O}\left(m \cdot n \cdot \log\left(2 + \frac{n \cdot \sqrt{\log(u_{\max})}}{m}\right)\right)$
1990	Cheriyen, Hagerup, Mehlhorn		$\mathcal{O}\left(\frac{n^3}{\log(n)}\right)$
1990	Alon		$\mathcal{O}\left(m \cdot n + \sqrt[3]{n^8} \cdot \log(n)\right)$
1992	King, Rao, Tarjan		$\mathcal{O}(m \cdot n + n^{2+\epsilon})$
1993	Philipps, Westbrook ^[1]		$\mathcal{O}\left(m \cdot n \cdot \log_m(n) + n^2 \cdot \log(n)^{2+\epsilon}\right)$
1994	King, Rao, Tarjan ^[2]		$\mathcal{O}\left(m \cdot n \cdot \log_{\frac{m}{n \cdot \log(n)}}(n)\right)$
1997	Goldberg, Rao ^[3]		$\mathcal{O}\left(\min\{\sqrt{m}, \sqrt[3]{n^2}\} \cdot m \cdot \log\left(\frac{n^2}{m}\right) \cdot \log(u_{\max})\right)$

Legende

- $n = |V|$ = „Anzahl der Knoten“
- $m = |E|$ = „Anzahl der Kanten“
- $u_{\max}$ = „Maximum der Kapazitätsfunktion u “

1. Steven J. Phillips, Jeffery Westbrook: *On-Line Load Balancing and Network Flow*. In *Algorithmica* Volume 21, Number 3, Juli 1998 245-261
 2. T. V. King, S. Rao, R. Tarjan: *A Faster Deterministic Maximum Flow Algorithm*. In *Journal of Algorithms*, Volume 17, Issue 3, November 1994 447-474
 3. Dávid Papp: *The Goldberg-Rao Algorithm for the Maximum Flow Problem* [\(PDF; 111 KB\)](#) (2006)

Nächste Vorlesung

Nächste Vorlesung

Montag 2. Juli, 08:30 (Hörsaal H01). Bis dann!