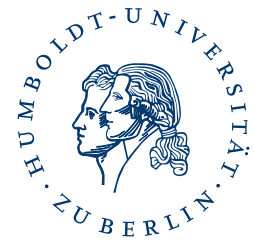
Graphalgorithmen

Johannes Köbler



Institut für Informatik
Humboldt-Universität zu Berlin

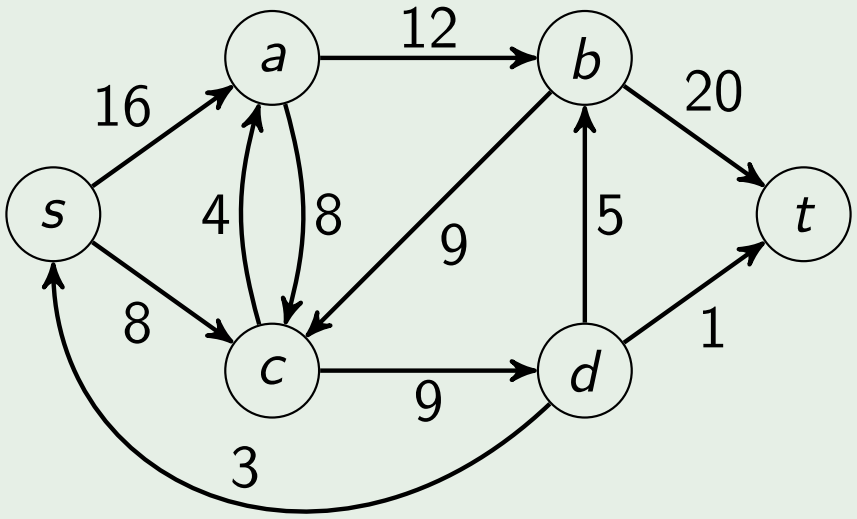
SS 2021

Flüsse in Netzwerken

Definition

- Ein Netzwerk $N = (V, E, s, t, c)$ besteht aus einem gerichteten Graphen $G = (V, E)$ mit einer Quelle $s \in V$ und einer Senke $t \in V$ sowie einer Kapazitätsfunktion $c : V \times V \to \mathbb{N}$
- Dabei muss jede Kante $(u, v) \in E$ eine Kapazität $c(u, v) > 0$ und jede Nichtkante $(u, v) \notin E$ muss die Kapazität $c(u, v) = 0$ haben

Beispiel. Die folgende Abbildung zeigt ein Netzwerk N.



Flüsse in Netzwerken

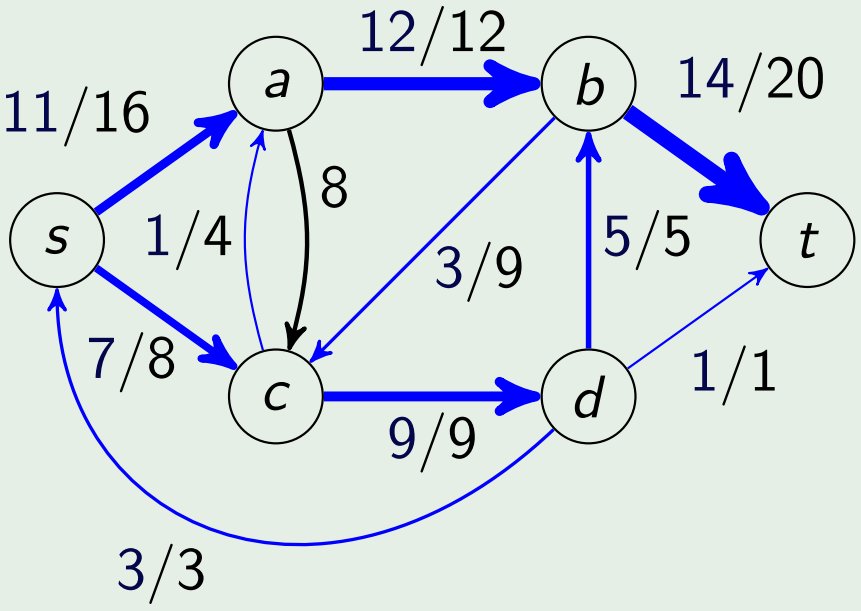
Definition. Sei $N = (V, E, s, t, c)$ ein Netzwerk.

- Ein Fluss in N ist eine Funktion $f : V \times V \to \mathbb{Z}$ mit
  - $f(u, v) \leq c(u, v)$ (Kapazitätsbedingung)
  - $f(u, v) = -f(v, u)$ (Antisymmetrie)
  - $\sum_{v \in V} f(u, v) = 0$ für alle $u \in V \setminus \{s, t\}$ (Kontinuität)
- Die Größe von f ist $|f| = \sum_{v \in V} f(s, v)$
- Der Fluss in den Knoten u ist $f^-(u) = \sum_{v \in V} \max\{0, f(v, u)\}$
- Der Fluss aus u ist $f^+(u) = \sum_{v \in V} \max\{0, f(u, v)\}$

- Die Antisymmetrie impliziert, dass $f(u, u) = 0$ für alle $u \in V$ gilt und $|f| = f^+(s) - f^-(s)$ ist
- Wir können also annehmen, dass $c(u, u) = 0$ für alle Knoten $u \in V$ gilt und somit $G = (V, E)$ schlingenfrei ist
- Die Kontinuität besagt, dass $f^+(u) = f^-(u)$ für alle $u \in V \setminus \{s, t\}$ gilt

Beispiel (Fortsetzung)

- Die Abbildung zeigt einen Fluss f in N:



u	s	a	b	c	d	t
$f^+(u)$	18	12	17	10	9	0
$f^-(u)$	3	12	17	10	9	15

- Dieser hat die Größe $|f| = \sum_{v \in V} f(s, v) = 11 + 7 - 3 = 15$

Der Ford-Fulkerson-Algorithmus

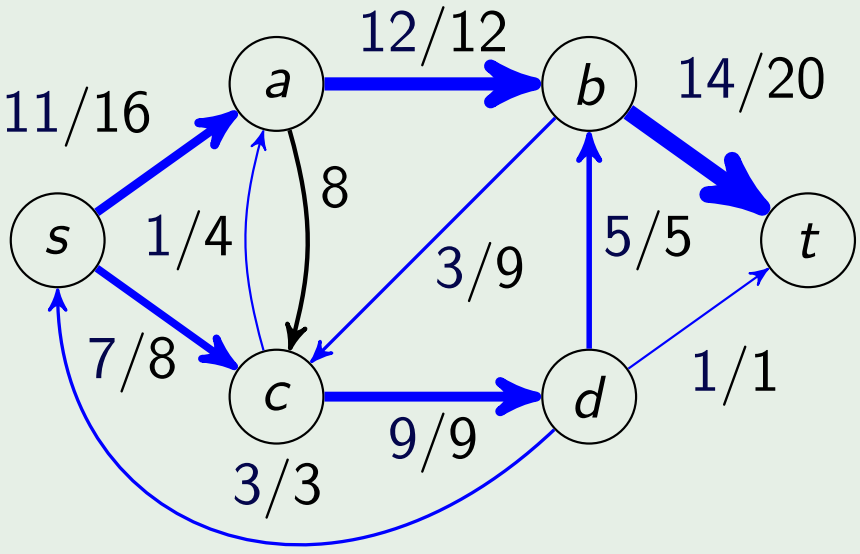
- Wie kann man für einen Fluss f in einem Netzwerk N entscheiden, ob er vergrößert werden kann?
- Diese Frage ist leicht zu beantworten, falls f auf $V \times V$ den Wert 0 hat
- In diesem Fall genügt es, in $G = (V, E)$ einen s-t-Pfad zu finden
- Andernfalls können wir ein Netzwerk N_f konstruieren, in dem sich der Nullfluss genau dann vergrößern lässt, wenn sich f in N vergrößern lässt

Definition

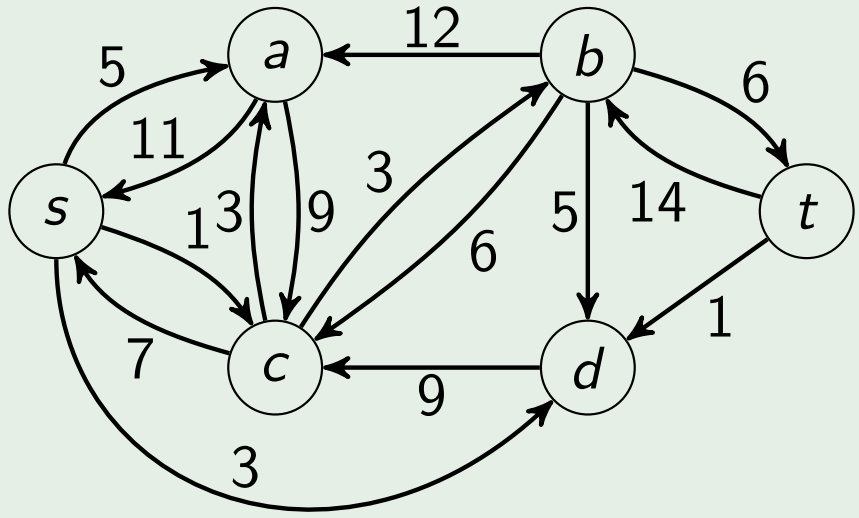
- Sei $N = (V, E, s, t, c)$ ein Netzwerk und sei f ein Fluss in N
- Das zugeordnete Restnetzwerk ist $N_f = (V, E_f, s, t, c_f)$ mit
  - den Kapazitäten $c_f(u, v) = c(u, v) - f(u, v)$ und
  - der Kantenmenge $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

Beispiel (Fortsetzung). Der Fluss f führt auf das folgende Restnetzwerk

Fluss f:



Restnetzwerk N_f:



Flüsse in Netzwerken

Definition. Sei $N = (V, E, s, t, c)$ ein Netzwerk.

- Dann heißt jeder s-t-Pfad P in (V, E) Zunahmepfad in N
- Die Kapazität von P in N ist

$$c(P) = \min\{c(u, v) : (u, v) \text{ liegt auf } P\}$$

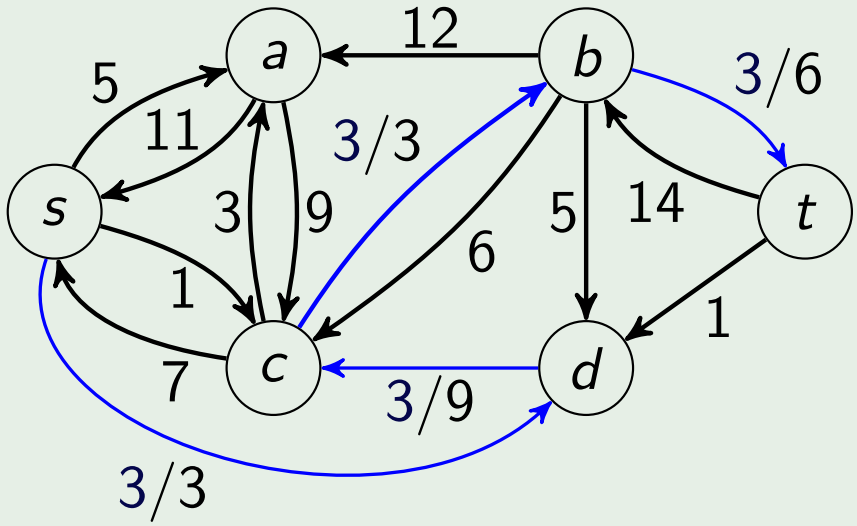
- und der Fluss durch P in N ist

$$f_P(u, v) = \begin{cases} c(P), & (u, v) \text{ liegt auf } P \\ -c(P), & (v, u) \text{ liegt auf } P \\ 0, & \textit{sonst} \end{cases}$$

- Es ist leicht zu sehen, dass f_P tatsächlich ein Fluss in N ist
- Ein Pfad $P = (u_0, \ldots, u_k)$ ist also ein Zunahmepfad in N, falls
  - $u_0 = s$ und $u_k = t$ ist
  - die Knoten $u_0, \ldots, u_k$ paarweise verschieden sind, und
  - $c(u_i, u_{i+1}) > 0$ für $i = 0, \ldots, k - 1$ gilt

Beispiel (Fortsetzung)

Die folgende Abbildung zeigt den Zunahmepfad $P = (s, d, c, b, t)$ in N_f mit der Kapazität $c_f(P) = 3$ und den zugehörigen Fluss f_P in N_f

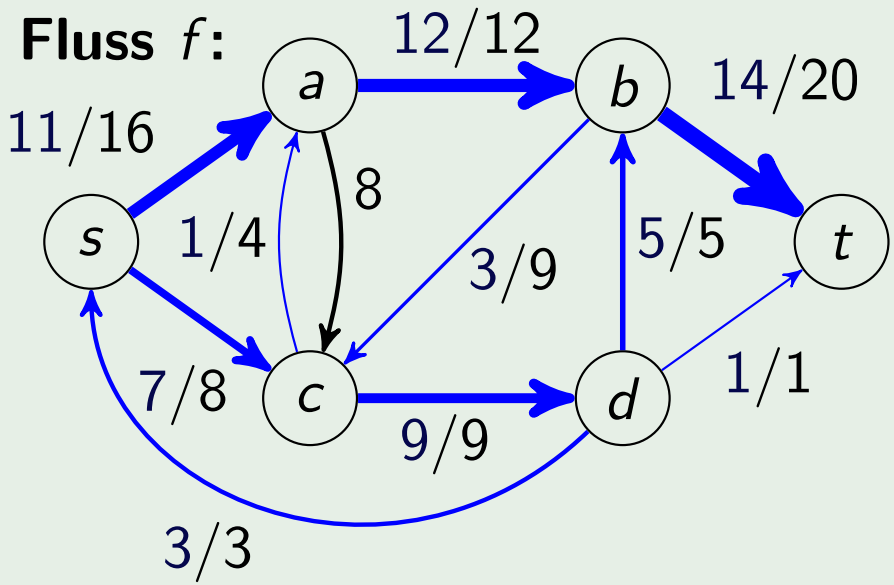


Durch Addition der beiden Flüsse f und f_P erhalten wir einen Fluss $f' = f + f_P$ in N der Größe $|f'| = |f| + |f_P| = |f| + c_f(P) > |f|$

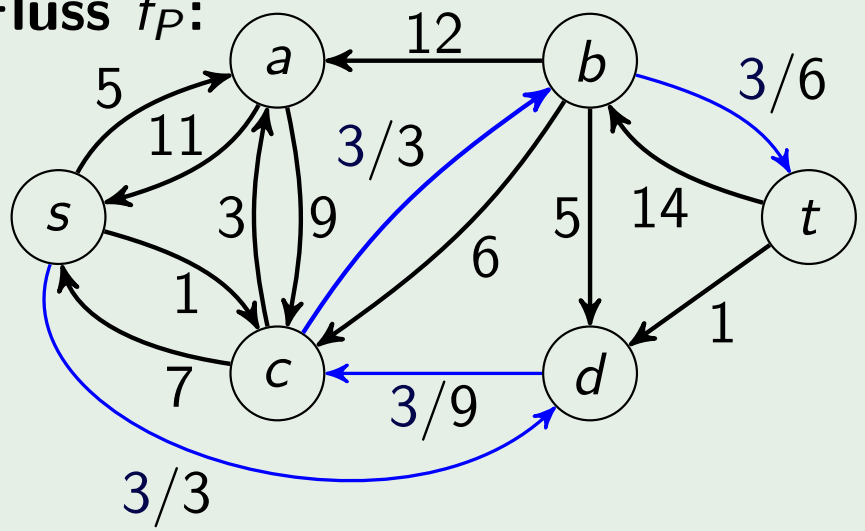
Flüsse in Netzwerken

Beispiel (Fortsetzung)

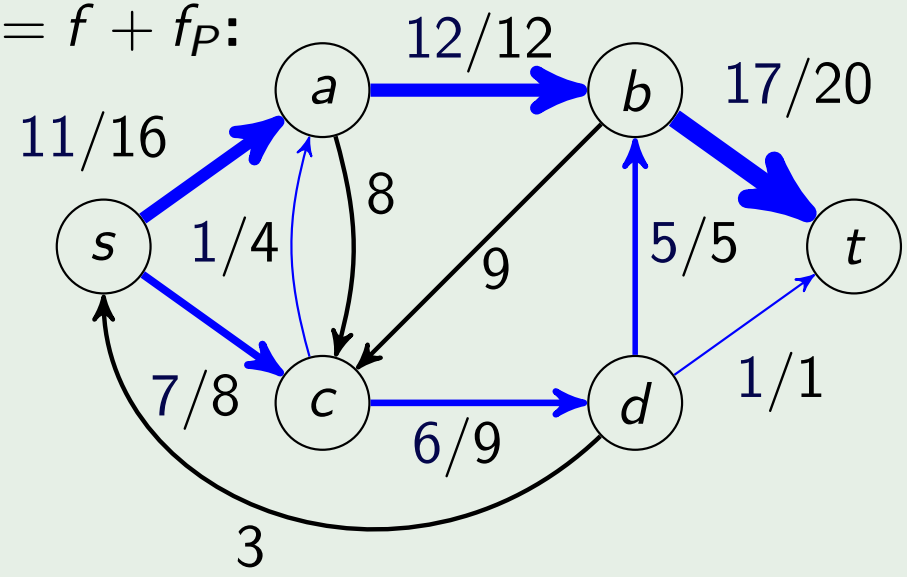
Fluss f:



Fluss f_P:



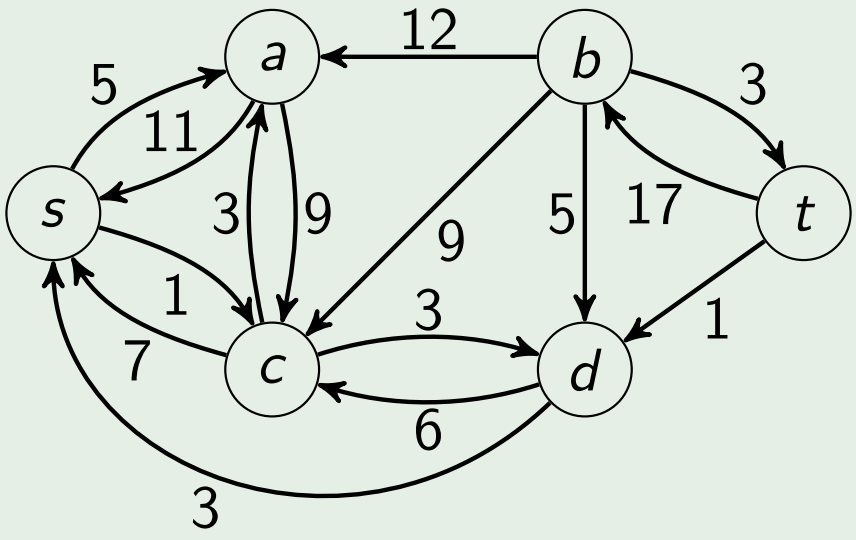
Fluss $f' = f + f_P$:



Algorithmus `Ford-Fulkerson`(V, E, s, t, c)

```
1  for all (u, v) ∈ E ∪ E^R do
2     f(u, v) := 0
3  while es gibt einen Zunahmepfad P in N_f do
4     f := f + f_P
```

Beispiel. Für den neuen Fluss erhalten wir nun folgendes Restnetzwerk:



In diesem existiert kein Zunahmepfad mehr ◁

Der Ford-Fulkerson-Algorithmus

Um zu zeigen, dass der Algorithmus von Ford-Fulkerson tatsächlich einen Maximalfluss berechnet, weisen wir nach, dass f maximale Größe in N hat, wenn im Restnetzwerk N_f kein Zunahmepfad existiert

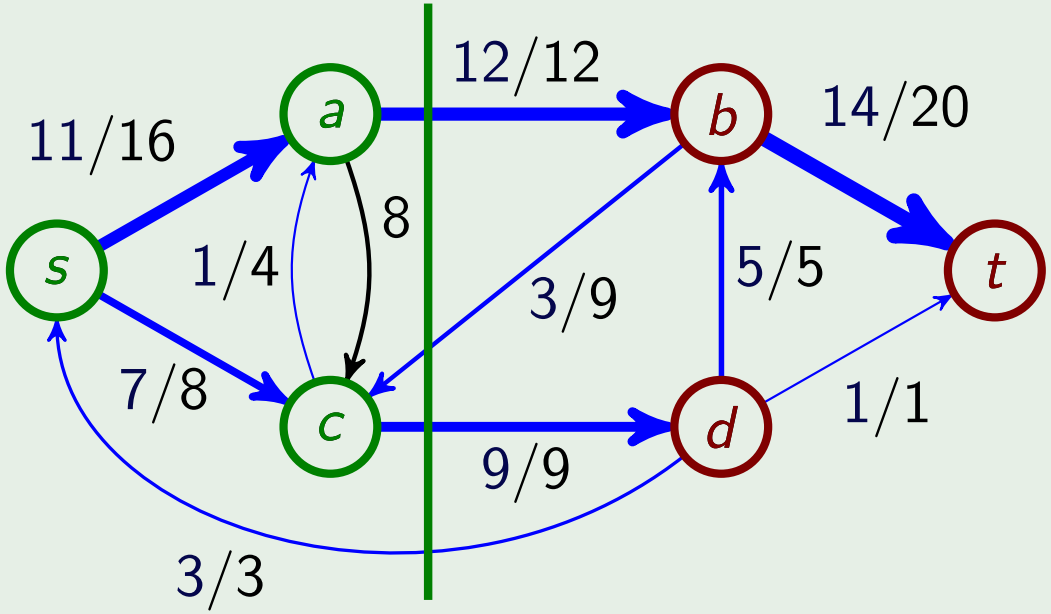
Definition. Sei $N = (V, E, s, t, c)$ ein Netzwerk und sei f ein Fluss in N.

- Eine Menge S mit $\emptyset \subsetneq S \subsetneq V$ heißt Schnitt durch N
- Der zugehörige Kantenschnitt ist $E(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$ (dieser wird oft auch einfach als Schnitt bezeichnet)
- Die Kapazität eines Schnittes S ist $c(S) = \sum_{e \in E(S)} c(e)$
- Der Fluss durch den Schnitt S ist $f(S) = \sum_{e \in E(S)} f(e)$
- Ist $u \in S$ und $v \notin S$, so wird S auch als u-v-Schnitt bezeichnet

Der Ford-Fulkerson-Algorithmus

Beispiel

- Betrachte folgenden s-t-Schnitt $S = \{s, a, c\}$ durch das Netzwerk N mit dem Fluss f:



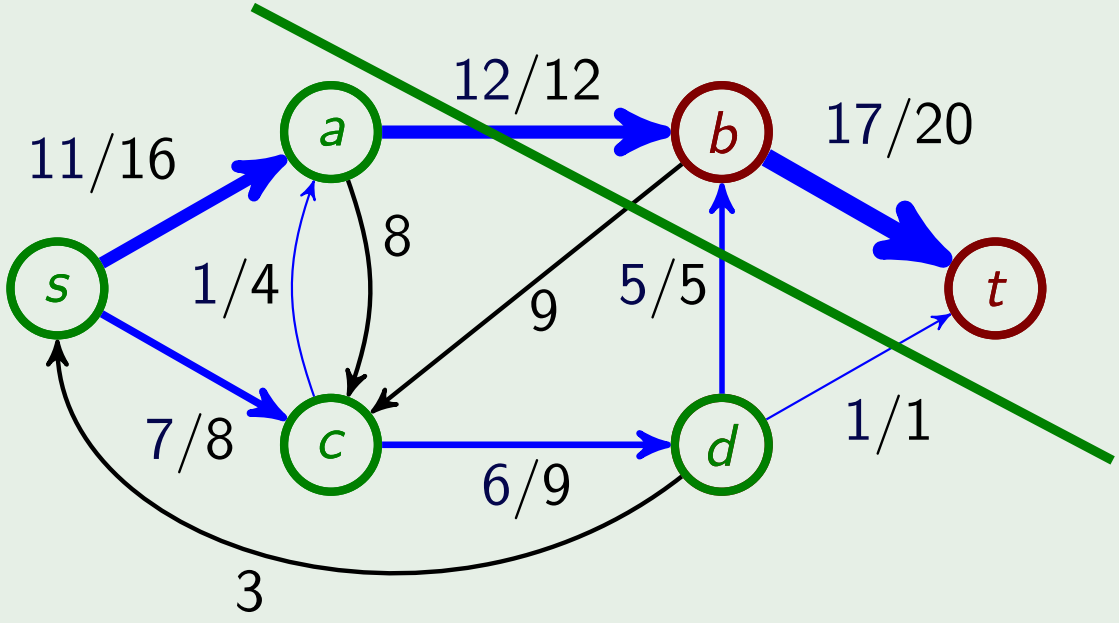
- Die Kapazität von S ist $c(S) = c(a, b) + c(c, d) = 12 + 9 = 21$ und die Größe des Flusses f durch den Schnitt S ist

$$f(S) = f(a, b) + f(c, b) + f(c, d) + f(s, d) = 12 - 3 + 9 - 3 = 15$$

Der Ford-Fulkerson-Algorithmus

Beispiel (Fortsetzung)

- Dagegen hat der s-t-Schnitt $S' = \{s, a, c, d\}$ durch das Netzwerk N mit dem Fluss f'



die Kapazität $c(S') = c(a,b) + c(d,b) + c(d,t) = 12 + 5 + 1 = 18$

- Diese stimmt mit der Größe des Flusses f' durch den Schnitt S' überein:

$$f'(S') = f'(a,b) + f'(d,b) + f'(d,t) = 12 + 5 + 1 = 18$$

◁

Der Ford-Fulkerson-Algorithmus

Lemma

Für jeden Fluss f in einem Netzwerk N und jeden s-t-Schnitt S durch N gilt $|f| = f(S) \leq c(S)$

Beweis.

- Wir zeigen zuerst die Ungleichung $f(S) \leq c(S)$
- Wegen $f(e) \leq c(e)$ für alle $e \in V \times V$ gilt

$$f(S) = \sum_{e \in E(S)} f(e) \leq \sum_{e \in E(S)} c(e) = c(S)$$

- Die Gleichheit $|f| = f(S)$ zeigen wir durch Induktion über $k = |S|$
  - Im Fall $k = 1$ (IA) ist $S = \{s\}$ und wegen $f(s, s) = 0$ folgt

$$|f| = \sum_{v} f(s, v) = \sum_{v \neq s} f(s, v) = f(S)$$

Der Ford-Fulkerson-Algorithmus

Beweis (Fortsetzung)

- Für den IS sei S ein s-t-Schnitt mit $|S| = k \geq 2$ und sei S' der Schnitt $S - \{w\}$, wobei wir $w \in S - \{s\}$ beliebig wählen
- Nach IV gilt dann $|f| = f(S')$ und wegen $S = S' \cup \{w\}$ folgt

$$f(S) = \sum_{u \in S, v \notin S} f(u,v) = \sum_{u \in S', v \notin S} f(u,v) + \sum_{v \notin S} f(w,v)$$

- Zudem folgt wegen $V \setminus S' = (V \setminus S) \cup \{w\}$

$$f(S') = \sum_{u \in S', v \notin S'} f(u,v) = \sum_{u \in S', v \notin S} f(u,v) + \sum_{u \in S'} f(u,w)$$

- Daher erhalten wir

$$f(S) - f(S') = \sum_{v \notin S} f(w,v) - \sum_{u \in S'} \underbrace{f(u,w)}_{=-f(w,u)} = \sum_{v \neq w} f(w,v) = 0$$

- Also gilt $f(S) = f(S') = |f|$

□

Der Ford-Fulkerson-Algorithmus

Satz (Max-Flow-Min-Cut-Theorem)

Für einen Fluss f in einem Netzwerk $N = (V, E, s, t, c)$ sind folgende Aussagen äquivalent:

1. f ist maximal, d.h. für jeden Fluss f' in N gilt $|f'| \leq |f|$
2. Im Restnetzwerk N_f existiert kein Zunahmepfad
3. Es gibt einen s-t-Schnitt S durch N mit $c(S) = |f|$

Beweis.

- Die Implikation 1 $\Rightarrow$ 2 ist klar, da die Existenz eines Zunahmepfads in N_f zu einer Vergrößerung von f führen würde
- Für die Implikation 2 $\Rightarrow$ 3 betrachten wir den Schnitt

$$S = \{u \in V \mid u \text{ ist in } N_f \text{ von } s \text{ aus erreichbar}\}$$

  - Dann gilt $t \notin S$ (da in N_f kein Zunahmepfad existiert)

Der Ford-Fulkerson-Algorithmus

Beweis (Fortsetzung)

- Für die Implikation (2) $\Rightarrow$ (3) betrachten wir den Schnitt

$$S = \{u \in V \mid u \text{ ist in } N_f \text{ von } s \text{ aus erreichbar}\}$$

  - Dann gilt $t \notin S$ (da in N_f kein Zunahmepfad existiert)
  - Zudem haben alle Kanten $e = (u, v) \in E(S)$ die Restkapazität $c_f(e) = c(e) - f(e) = 0$ (sonst wäre mit u auch v in S enthalten)
  - Daher folgt

$$|f| = f(S) = \sum_{e \in E(S)} f(e) = \sum_{e \in E(S)} c(e) = c(S)$$

- Die Implikation (3) $\Rightarrow$ (1) folgt direkt aus obigem Lemma, da jeder Fluss f' in N im Fall $c(S) = |f|$ einen Wert $|f'| = f'(S) \leq c(S) = |f|$ hat $\square$

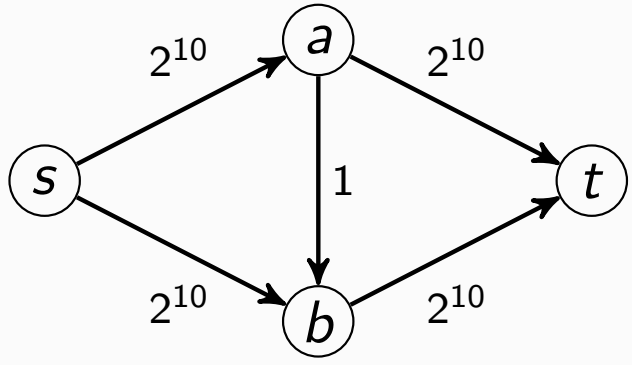
Das Max-Flow-Min-Cut-Theorem gilt auch für Netzwerke mit beliebigen reellen Kapazitäten $c(e) \geq 0$

Der Ford-Fulkerson-Algorithmus

- Ist $c_0 = c(S)$ die Kapazität des Schnittes $S = \{s\}$, so durchläuft der Ford-Fulkerson-Algorithmus die while-Schleife höchstens c_0-mal, da sich der aktuelle Fluss in jedem Durchlauf um mindestens 1 erhöht
- Bei jedem Durchlauf ist zuerst das Restnetzwerk N_f und danach ein Zunahmepfad in N_f zu berechnen
  - Die Berechnung des Zunahmepfads P kann durch Breitensuche in Zeit $\mathcal{O}(n + m)$ erfolgen
  - Da sich das Restnetzwerk nur entlang von P ändert, kann es in Zeit $\mathcal{O}(n)$ aktualisiert werden
- Jeder Durchlauf benötigt also Zeit $\mathcal{O}(n + m)$, was auf eine Gesamtlaufzeit von $\mathcal{O}(c_0(n + m))$ führt
- Da der Wert von c_0 jedoch exponentiell in der Länge der Eingabe (also der Beschreibung des Netzwerkes N) sein kann, ergibt dies keine polynomiell beschränkte Laufzeit
- Bei Netzwerken mit reellen Kapazitäten kann Ford-Fulkerson sogar unendlich lange laufen (siehe Übungen)

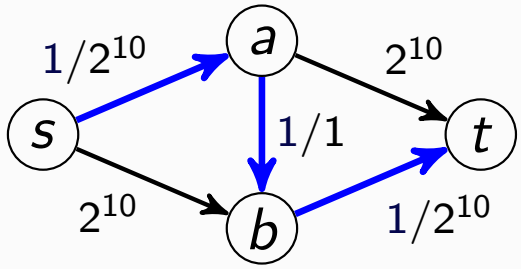
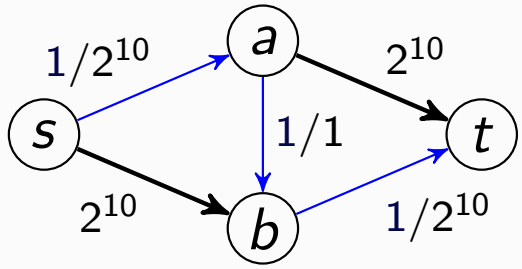
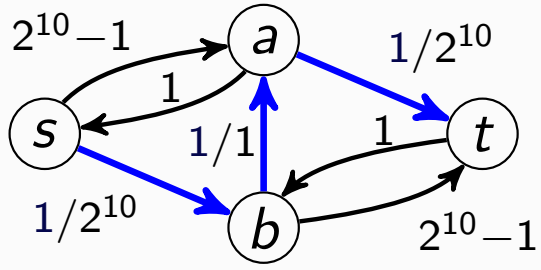
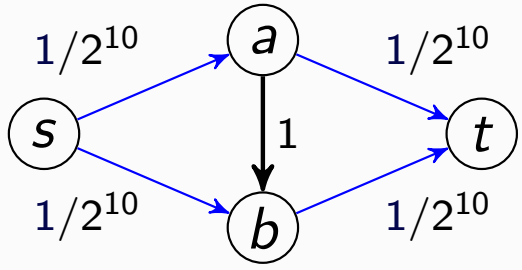
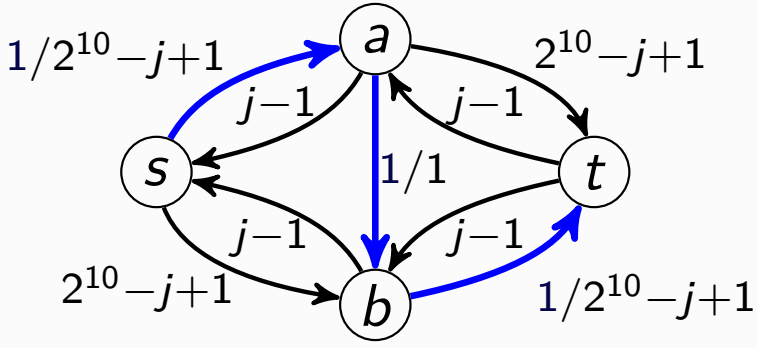
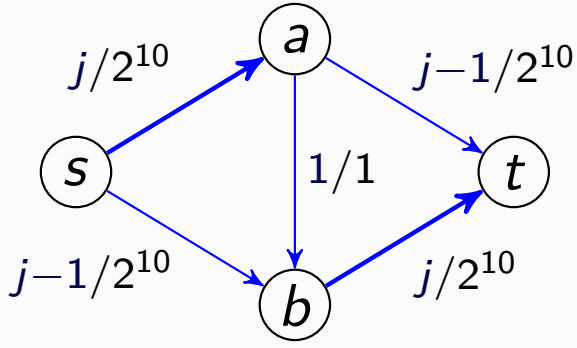
Der Ford-Fulkerson-Algorithmus

- Bei nebenstehendem Netzwerk benötigt Ford-Fulkerson zur Bestimmung des Maximalflusses abhängig von der Wahl der Zunahmepfade zwischen 2 und 2^{11} Schleifendurchläufe

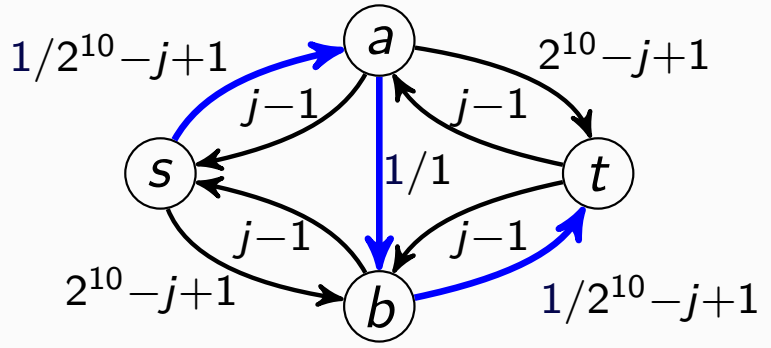
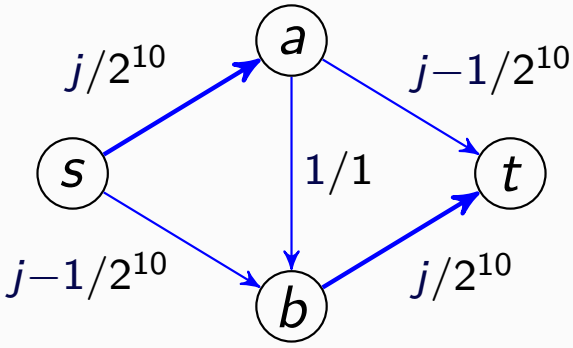
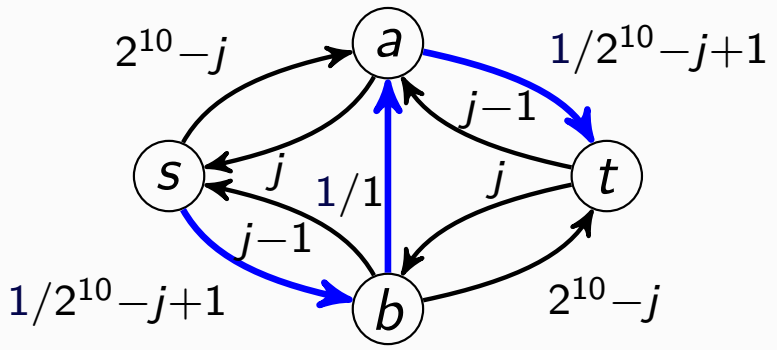
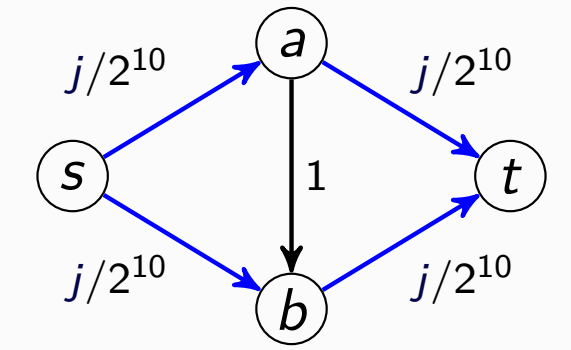
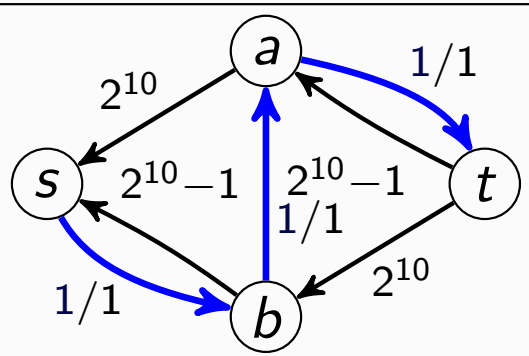
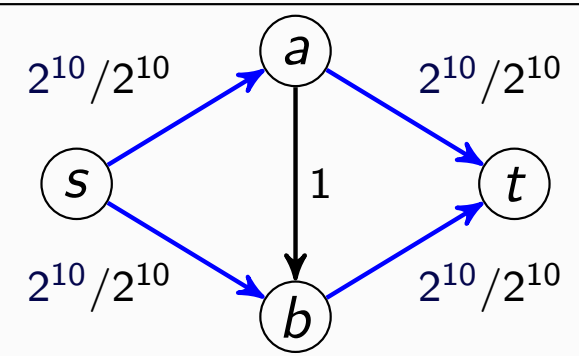


- Im günstigsten Fall wird nämlich ausgehend vom Nullfluss f_0 zuerst der Zunahmepfad $P_1 = (s, a, t)$ mit der Kapazität 2^{10} und dann im Restnetzwerk N_{f_1} der Pfad $P_2 = (s, b, t)$ mit der Kapazität 2^{10} gewählt
- Im ungünstigsten Fall werden abwechselnd die beiden Zunahmepfade $P_1 = (s, a, b, t)$ und $P_2 = (s, b, a, t)$ (also $P_i = P_1$ für ungerades i und $P_i = P_2$ für gerades i) mit der Kapazität 1 gewählt. Dies führt auf insgesamt 2^{11} Schleifendurchläufe (siehe folgende Tabelle)

Der Ford-Fulkerson-Algorithmus

i	Fluss f_{P_i} in N_{f_i}	neuer Fluss f_{i+1} in N
1		
2		
$\vdots$		
$2j-1$, $1 < j \leq 2^{10}$		

Der Ford-Fulkerson-Algorithmus

$2j-1$, $1 < j \leq 2^{10}$		
$2j$, $1 < j < 2^{10}$		
$\vdots$		
2^{11}		

Der Ford-Fulkerson-Algorithmus

Nicht nur in diesem Beispiel lässt sich die exponentielle Laufzeit wie folgt vermeiden:

- Man betrachtet nur Zunahmepfade mit einer geeignet gewählten Mindestkapazität
  - Dies führt auf eine Laufzeit, die polynomiell in n, m und $\log c_0$ ist
- Man bestimmt in jeder Iteration einen kürzesten Zunahmepfad im Restnetzwerk mittels Breitensuche in Zeit $\mathcal{O}(n+m)$
  - Diese Vorgehensweise führt auf den Edmonds-Karp-Algorithmus, der eine Laufzeit von $\mathcal{O}(nm^2)$ hat (unabhängig von den Kapazitäten)
  - Diesen werden wir als nächstes betrachten

Der Ford-Fulkerson-Algorithmus

- Dinitz hatte die Idee, in jeder Iteration einen blockierenden Fluss g in N_f zu berechnen
  - Dieser benutzt nur Kanten, die auf einem kürzesten s-t-Pfad in N_f liegen, und sättigt auf jedem kürzesten s-t-Pfad mindestens eine Kante e (d.h. $g(e) = c_f(e)$), die dann in der nächsten Iteration fehlt
  - Da die Länge der kürzesten s-t-Pfade im Restnetzwerk in jeder Iteration um mindestens eins zunimmt, liegt nach spätestens $n - 1$ Iterationen ein maximaler Fluss vor
  - Dinitz hat gezeigt, dass der Fluss g in Zeit $\mathcal{O}(nm)$ bestimmt werden kann, was auf eine Laufzeit von $\mathcal{O}(n^2 m)$ führt
- Malhotra, Kumar und Maheswari fanden später einen Ansatz, den Fluss g in Zeit $\mathcal{O}(n^2)$ zu berechnen, wodurch sich die Laufzeit auf $\mathcal{O}(n^3)$ verbessern ließ

Der Edmonds-Karp-Algorithmus

- Der Edmonds-Karp-Algorithmus ist eine spezielle Form von Ford-Fulkerson, die möglichst kurze Zunahmepfade benutzt
- Diese können mittels Breitensuche bestimmt werden

Algorithmus `Edmonds-Karp`(V, E, s, t, c)

```
1 for all (u, v) ∈ E ∪ E^R do
2   f(u, v) := 0
3 while P := zunahmepfad(f) ≠ ⊥ do
4   addierepfad(f, P)
```

Prozedur `addierepfad`(f, P)

```
1 for all e ∈ P do
2   f(e) := f(e) + c_f(P)
3   f(e^R) := f(e^R) − c_f(P)
```

Der Edmonds-Karp-Algorithmus

Prozedur `zunahmepfad(`f`)`

```
1  for all v ∈ V do
2    parent(v) := ⊥
3  Q := (s)
4  while Q ≠ () ∧ parent(t) = ⊥ do
5    u := dequeue(Q)
6    for all e = (u, v) ∈ E ∪ E^R do
7      if c(e) − f(e) > 0 ∧ parent(v) = ⊥ then
8        c'(e) := c(e) − f(e)
9        parent(v) := u
10       enqueue(Q, v)
11 if parent(t) = ⊥ then
12   P := ⊥
13 else
14   P := parent-Pfad von s nach t
15   c_f(P) := min{c'(e) | e ∈ P}
16 return P
```

Der Edmonds-Karp-Algorithmus

- Die Prozedur `zunahmepfad(`f`)` berechnet im Restnetzwerk N_f einen (gerichteten) s-t-Pfad P kürzester Länge, sofern ein s-t-Pfad existiert
- Dies ist genau dann der Fall, wenn die while-Schleife mit $\texttt{parent}(t) \neq \perp$ abbricht
- Der gefundene Zunahmepfad $P = (u_\ell, \ldots, u_0)$ lässt sich dann ausgehend von $u_0 = t$ mittels `parent` zurückverfolgen:

$$u_i = \begin{cases} t, & i = 0, \\ \texttt{parent}(u_{i-1}), & i > 0 \text{ und } u_{i-1} \neq s \end{cases}$$

  wobei $\ell = \min\{i \geq 1 \mid u_i = s\}$ ist
- Dann ist $u_\ell = s$ und P ein s-t-Pfad, den wir als den parent-Pfad von s nach t bezeichnen

Der Edmonds-Karp-Algorithmus

Satz

Der Edmonds-Karp-Algorithmus durchläuft die while-Schleife höchstens $(nm/2)$-mal und hat somit eine Laufzeit von $O(nm^2)$

Beweis.

- Sei k die Anzahl der Schleifendurchläufe und seien $P_1, \ldots, P_k$ die Zunahmepfade, die der Algorithmus bei Eingabe N berechnet
- Es gilt also $f_{i+1} = f_i + f_{P_{i+1}}$, wobei f_0 der triviale Nullfluss und P_{i+1} der im Restnetzwerk N_{f_i} berechnete Zunahmepfad ist
- Eine Kante e auf P_{i+1} heißt kritisch für P_{i+1}, falls der Fluss $f_{P_{i+1}}$ in N_{f_i} die Kante e sättigt, d.h. $f_{P_{i+1}}(e) = c_{f_i}(e)$
- Eine kritische Kante e für P_{i+1} ist wegen

$$c_{f_{i+1}}(e) = c(e) - f_{i+1}(e) = c(e) - (f_i + f_{P_{i+1}}) = c_{f_i}(e) - f_{P_{i+1}}(e) = 0$$

nicht in $N_{f_{i+1}}$ enthalten, wohl aber die Kante e^R:

$$c_{f_{i+1}}(e^R) = c(e^R) - f_{i+1}(e^R) = c(e^R) + f_{i+1}(e) = c(e^R) + c(e) > 0$$

Der Edmonds-Karp-Algorithmus

Beweis (Fortsetzung)

- Sei $d_i(u, v)$ die minimale Länge eines Pfades von u nach v im Restnetzwerk N_{f_i} und sei $\ell_{i+1} = d_i(s, t)$ die Länge von P_{i+1}
- Für den Rest des Beweises benötigen wir folgende drei Behauptungen

Behauptung 1.

Für jeden Knoten $u \in V$ gilt $d_i(s, u) \leq d_{i+1}(s, u)$ und $d_i(u, t) \leq d_{i+1}(u, t)$

Behauptung 2. Sei $e = (u, v)$ eine Kante auf dem Pfad P_{i+1}.

Falls e^R für ein $j > i$ auf dem Pfad P_{j+1} liegt, dann ist $\ell_{j+1} \geq \ell_{i+1} + 2$

Behauptung 3.

Seien $P_{i_1}, \ldots, P_{i_h}$ mit $1 \leq i_1 < \cdots < i_h \leq k$ die Pfade, für die e oder e^R kritisch sind. Dann ist $h \leq n/2$.

Der Edmonds-Karp-Algorithmus

Behauptung 1.

Für jeden Knoten $u \in V$ gilt $d_i(s,u) \leq d_{i+1}(s,u)$ und $d_i(u,t) \leq d_{i+1}(u,t)$

Beweis von Behauptung 1.

- Hierzu beweisen wir für jeden kürzesten Pfad $P = (u_0, \ldots, u_\ell)$ von $u_0 = s$ nach $u_\ell = u$ in $N_{f_{i+1}}$ (d.h. $d_{i+1}(s,u) = \ell$) die Ungleichungen

  $$d_i(s,u_h) \leq d_i(s,u_{h-1}) + 1 \text{ für } h = 1, \ldots, \ell,$$

  die $d_i(s,u) \leq \ell$ implizieren
- Falls die Kante $e = (u_{h-1}, u_h)$ in N_{f_i} enthalten ist, ist nichts zu zeigen
- Andernfalls muss $f_{i+1}(e) \neq f_i(e)$ sein, d.h. e oder e^R liegen auf P_{i+1}
- Da e nicht in N_{f_i} ist, muss $e^R = (u_h, u_{h-1})$ auf P_{i+1} liegen
- Da P_{i+1} ein kürzester Pfad von s nach t in N_{f_i} ist, folgt $d_i(s,u_{h-1}) = d_i(s,u_h) + 1$, was $d_i(s,u_h) = d_i(s,u_{h-1}) - 1 \leq d_i(s,u_{h-1}) + 1$ impliziert
- Vollkommen analog lässt sich $d_i(u,t) \leq d_{i+1}(u,t)$ zeigen □

Behauptung 2. Sei $e = (u, v)$ eine Kante auf dem Pfad P_{i+1}.

Falls e^R für ein $j > i$ auf dem Pfad P_{j+1} liegt, dann ist $\ell_{j+1} \geq \ell_{i+1} + 2$

Beweis von Behauptung 2.

- Da P_{i+1} ein kürzester s-t-Pfad der Länge $\ell_{i+1} = d_i(s, t)$ in N_{f_i} und P_{j+1} ein kürzester s-t-Pfad der Länge $\ell_{j+1} = d_j(s, t)$ in N_{f_j} ist, folgt dies direkt aus Behauptung 1:

$$d_j(s,t) = \underbrace{d_j(s,v)}_{\geq d_i(s,v)} + \underbrace{d_j(u,t)}_{\geq d_i(u,t)} + 1 \geq \underbrace{d_i(s,v)}_{d_i(s,u)+1} + \underbrace{d_i(u,t)}_{d_i(v,t)+1} + 1 = d_i(s,t) + 2$$

□

Der Edmonds-Karp-Algorithmus

Behauptung 3.

Seien $P_{i_1}, \ldots, P_{i_h}$ mit $1 \leq i_1 < \cdots < i_h \leq k$ die Pfade, für die e oder e^R kritisch sind. Dann ist $h \leq n/2$.

Beweis von Behauptung 3.

- Wir zeigen zuerst, dass $\ell_{i_{j+1}} \geq \ell_{i_j} + 2$ für $j = 1, \ldots, h-1$ gilt
- Falls $e' \in \{e, e^R\}$ für ein $j \in \{1, \ldots, h-1\}$ kritisch für den Pfad P_{i_j} ist, dann fehlt e' im Restnetzwerk $N_{f_{i_j}}$
- Daher kann e' nur dann eine kritische Kante für den Pfad $P_{i_{j+1}}$ sein, wenn e'^R auf einem Pfad P_i mit $i_j < i \leq i_{j+1}$ liegt
- Dies gilt natürlich erst recht, wenn die Kante e'^R für $P_{i_{j+1}}$ kritisch ist
- Mit Behauptung 1 und Behauptung 2 folgt also $\ell_{i_{j+1}} \geq \ell_i \geq \ell_{i_j} + 2$
- Daher ist

$$n - 1 \geq \ell_{i_h} \geq \ell_{i_1} + 2(h-1) \geq 1 + 2(h-1) = 2h - 1,$$

was $h \leq n/2$ impliziert □

Beweis des Satzes (Schluss)

- Nach Behauptung 3 sind e und e^R für jede Kante $e \in E$ zusammen höchstens auf $n/2$ vielen Pfaden P_i kritisch
- Da $E \cup E^R$ höchstens m Kantenpaare der Form $\{e, e^R\}$ enthält, können die k Pfade $P_1, \ldots, P_k$ köchstens $mn/2$ kritische Kanten enthalten
- Da aber auf jedem Pfad P_i mindestens eine Kante kritisch ist, muss $k \leq mn/2$ sein □

Man beachte, dass der Beweis auch bei Netzwerken mit reellen Kapazitäten seine Gültigkeit behält

Der Algorithmus von Dinitz

- In den Übungen wird gezeigt, dass sich in jedem Netzwerk N ein maximaler Fluss durch Addition von höchstens m Zunahmepfaden konstruieren lässt
- Es ist aber nicht bekannt, ob sich solche Pfade in Zeit $O(m)$ bestimmen lassen
- Wenn ja, würde dies auf eine Gesamtlaufzeit von $O(m^2)$ führen
- Für dichte Netzwerke (d.h. $m = \Theta(n^2)$) hat der Algorithmus von Dinitz die gleiche Laufzeit $O(n^2 m) = O(n^4)$ und die verbesserte Version ist mit $O(n^3)$ in diesem Fall sogar noch schneller

Der Algorithmus von Dinitz

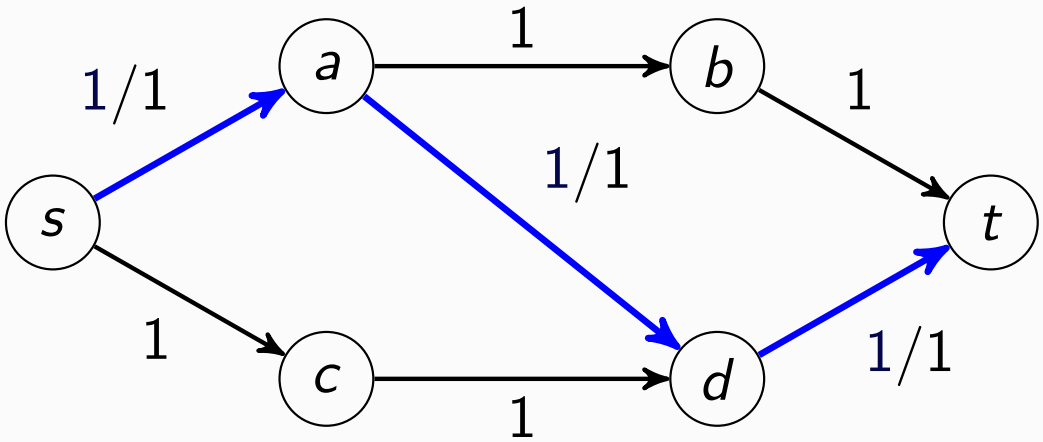
- Die Analyse der Laufzeit des Edmonds-Karp-Algorithmus benutzt die Tatsache, dass der Fluss durch den Zunahmepfad P_{i+1}, der in jedem Schleifendurchlauf auf den aktuellen Fluss f_i addiert wird, eine Kante auf *mindestens einem* kürzesten Pfad im Restnetzwerk N_{f_i} sättigt
- Dies hat zur Folge, dass nicht mehr als $nm/2$ Zunahmepfade P_i benötigt werden, um einen maximalen Fluss zu erhalten
- Dagegen addiert der Algorithmus von Dinitz in jedem Schleifendurchlauf auf den aktuellen Fluss f einen Fluss g, der auf *jedem* kürzesten Pfad im Restnetzwerk N_f mindestens eine Kante sättigt
- Wir werden sehen, dass maximal $n-1$ solche Flüsse g_i benötigt werden

Definition

- Ein Fluss g in einem Netzwerk $N = (V, E, s, t, c)$ ***sättigt*** eine Kante $e \in E$, falls $g(e) = c(e)$ ist
- g heißt ***blockierend***, falls g mindestens eine Kante auf jedem Pfad P von s nach t sättigt

Der Algorithmus von Dinitz

- Nach dem Max-Flow-Min-Cut-Theorem gibt es zu jedem maximalen Fluss f einen s-t-Schnitt S, so dass alle Kanten in $E(S)$ gesättigt sind
- Da jeder Pfad von s nach t mindestens eine Kante in $E(S)$ enthalten muss, ist jeder maximale Fluss auch blockierend
- Für die Umkehrung gibt es jedoch einfache Gegenbeispiele, wie etwa



- Ein blockierender Fluss muss also nicht unbedingt maximal sein
- Tatsächlich ist g genau dann ein blockierender Fluss in N, wenn es im Restnetzwerk N_g keinen Zunahmepfad gibt, der nur aus *Vorwärtskanten* $e \in E$ mit $g(e) < c(e)$ besteht

Der Algorithmus von Dinitz

- Der Algorithmus von Dinitz berechnet anstelle eines kürzesten Zunahmepfades P im aktuellen Restnetzwerk N_f einen blockierenden Fluss g im Schichtnetzwerk N'_f
- Dieses enthält nur diejenigen Kanten von N_f, die auf einem kürzesten Pfad mit Startknoten s liegen
- Zudem werden aus N'_f alle Knoten $u \neq t$ entfernt, die einen Abstand $d(s,u) \geq d(s,t)$ in N_f haben
- Der Name rührt daher, dass jeder Knoten in N'_f einer Schicht S_j zugeordnet wird

Der Algorithmus von Dinitz

Definition

- Sei $N = (V, E, s, t, c)$ ein Netzwerk und bezeichne $d(x, y)$ die Länge eines kürzesten Pfades von x nach y in N
- Das zugeordnete ***Schichtnetzwerk*** ist $N' = (V', E', s, t, c')$ mit
  - der Knotenmenge $V' = S_0 \cup \cdots \cup S_\ell$
  - der Kantenmenge $E' = \bigcup_{j=1}^{\ell} \{(u, v) \in E \mid u \in S_{j-1} \wedge v \in S_j\}$ und
  - der Kapazitätsfunktion

$$c'(e) = \begin{cases} c(e), & e \in E', \\ 0, & \text{sonst}, \end{cases}$$

wobei

$$S_j = \begin{cases} \{u \in V \mid d(s, u) = j\}, & 0 \leq j \leq \ell - 1 \\ \{t\}, & j = \ell \end{cases}$$

und $\ell = 1 + \max\{d(s, u) < d(s, t) \mid u \in V\}$ ist

Der Algorithmus von Dinitz

Algorithmus `Dinitz`(N), $N = (V, E, s, t, c)$

```
1 for all (u, v) ∈ E ∪ E^R do
2   f(u, v) := 0
3 while S := schichtnetzwerk(N, f) ≠ ⊥ do
4   f := f + blockfluss(S)
```

- Das zum Restnetzwerk $N_f = (V, E_f, s, t, c_f)$ gehörige Schichtnetzwerk $N'_f = (V', E'_f, s, t, c'_f)$ wird von der Prozedur `schichtnetzwerk`(N, f) in Zeit $O(n + m)$ berechnet
- Für die Berechnung eines blockierenden Flusses g im Schichtnetzwerk N'_f werden wir zwei Algorithmen betrachten
- Eine Prozedur `blockfluss1`, deren Laufzeit durch $O(nm)$ und eine Prozedur `blockfluss2`, deren Laufzeit durch $O(n^2)$ beschränkt ist
- Wir beschreiben zuerst die Prozedur `schichtnetzwerk`

Der Algorithmus von Dinitz

- Diese führt in N_f eine modifizierte Breitensuche mit Startknoten s durch und speichert dabei in der Menge E' nicht nur alle Baumkanten, sondern zusätzlich alle Querkanten (u, v) (d.h. u und v liegen nicht auf einem gemeinsamen parent-Pfad), die auf einem kürzesten Weg von s zu v liegen
- Die Suche bricht ab, sobald t am Kopf der Schlange erscheint oder alle von s aus erreichbaren Knoten abgearbeitet sind
- Falls t erreicht wird, werden außer der Senke t alle Knoten u, die in N_f einen Abstand $d(s, u) < d(s, t)$ von der Quelle s haben, in der Menge V' zusammengefasst
- Zudem werden alle Kanten aus E' wieder entfernt, die nicht zwischen zwei Knoten aus V' verlaufen
- Wird t dagegen nicht erreicht, so existiert in N_f (und damit in N'_f) kein (blockierender) Fluss g mit $|g| > 0$ und somit auch kein Zunahmepfad in N_f, d.h. f ist bereits maximal

Prozedur schichtnetzwerk(N, f)

```
1  E' := ∅
2  for all v ∈ V do niv(v) := n
3  niv(s) := 0; Q := (s)
4  while Q ≠ () ∧ head(Q) ≠ t do
5     u := dequeue(Q)
6     for all e = (u, v) ∈ E ∪ E^R do
7        if c(e) − f(e) > 0 ∧ niv(v) > niv(u) then
8           E' := E' ∪ {e}
9           c'(e) := c(e) − f(e)
10          if niv(v) > niv(u) + 1 then
11             niv(v) := niv(u) + 1
12             enqueue(Q, v)
13 if head(Q) = t then
14    V' := {v ∈ V | niv(v) < niv(t)} ∪ {t}
15    E' := E' ∩ (V' × V')
16    return (V', E', s, t, c')
17 else return ⊥
```

Der Algorithmus von Dinitz

- Die Laufzeitschranke $O(n+m)$ für die Prozedur `schichtnetzwerk` folgt aus der Tatsache, dass jede Kante in $E \cup E^R$ höchstens einmal besucht wird und jeder Besuch mit einem konstanten Zeitaufwand verbunden ist
- Nun kommen wir zur Prozedur `blockfluss1`, die einen blockierenden Fluss g in einem gegebenen Schichtnetzwerk S berechnet
- Beginnend mit dem Nullfluss g bestimmt diese in der repeat-Schleife
  - mittels Tiefensuche einen s-t-Pfad P in S
  - addiert den Fluss f_P durch P in S zum aktuellen Fluss g hinzu
  - aktualisiert die Kapazitäten aller Kanten e auf dem Pfad P und
  - entfernt aus S die von g gesättigten Kanten
- Der gefundene Pfad P lässt sich hierbei direkt aus dem Inhalt des Kellers K rekonstruieren, weshalb wir ihn als ***K-Pfad*** bezeichnen
- Man beachte, dass die Kapazitäten der auf P liegenden Kanten nur in Vorwärtsrichtung und nicht wie bei Ford-Fulkerson und Edmonds-Karp auch in Rückwärtsrichtung angepasst werden

Der Algorithmus von Dinitz

- Falls die Tiefensuche in einem Knoten $u \neq s$ in einer Sackgasse endet (weil E' keine von u aus weiterführenden Kanten enthält), wird die zuletzt besuchte Kante (u', u) ebenfalls aus E' entfernt und die Tiefensuche vom Startpunkt u' dieser Kante fortgesetzt (backtracking)
- Die Prozedur `blockfluss1` bricht ab, sobald alle Kanten mit Startknoten s aus E' entfernt wurden und somit in (V', E') keine Pfade mehr von s nach t existieren (d.h. g ist ein blockierender Fluss in S)
- Die Laufzeitschranke $O(nm)$ für `blockfluss1` folgt aus der Tatsache, dass sich die Anzahl der aus E' entfernten Kanten nach spätestens n Schleifendurchläufen um 1 erhöht

Prozedur `blockfluss1`(S), $S = (V', E', s, t, c')$

```
1  for all e ∈ E' ∪ E'^R do g(e) := 0
2  u := s; K := (s); done := false
3  repeat
4     if ∃e = (u, v) ∈ E' then
5        push(K, v); u := v
6     elsif u = t then
7        P := K-Pfad von s nach t
8        c'(P) := min{c'(e) | e ∈ P}
9        for all e ∈ P do
10          g(e) := g(e) + c'(P); g(e^R) := −g(e); c'(e) := c'(e)−c'(P)
11          if c'(e) = 0 then E' := E' \ {e}
12       K := (s); u := s
13    elsif u ≠ s then \\ backtracking
14       pop(K); u' := top(K); E' := E' \ {(u', u)}; u := u'
15    else done := true
16 until done
17 return g
```

Der Algorithmus von Dinitz

Satz

Der Algorithmus von Dinitz durchläuft die while-Schleife höchstens $(n-1)$-mal

Beweis.

- Sei f_0 der Nullfluss in N und seien $g_1, \ldots, g_k$ die blockierenden Flüsse, die der Dinitz-Algorithmus der Reihe nach berechnet, d.h. $f_{i+1} = f_i + g_{i+1}$
- Zudem sei $d_i(u, v)$ die minimale Länge eines Pfades von u nach v im Restnetzwerk N_{f_i} und sei $\delta_i = d_i(s, t)$
- Wir zeigen, dass $\delta_i < \delta_{i+1}$ für $i = 1, \ldots, k-1$ gilt
- Da $\delta_1 \geq 1$ und $\delta_k \leq n-1$ ist, folgt $k \leq n-1$

Der Algorithmus von Dinitz

Beweis (Fortsetzung)

- Sei $P = (u_0, \ldots, u_\ell)$ ein kürzester Pfad von $u_0 = s$ nach $u_\ell = u$ in $N_{f_{i+1}}$ (d.h. es gilt $d_{i+1}(s, u_h) = h$ für $h = 1, \ldots, \ell$)
- Wir beweisen für $h = 1, \ldots, \ell$ die folgenden (Un)gleichungen:

$$d_i(s, u_h) \leq d_i(s, u_{h-1}) + 1, \text{ falls } (u_{h-1}, u_h) \in E_{f_i} \quad (1)$$

$$d_i(s, u_h) = d_i(s, u_{h-1}) - 1, \text{ falls } (u_{h-1}, u_h) \notin E_{f_i} \quad (2)$$

- Es ist klar, dass (1) gilt
- Falls die Kante $e = (u_{h-1}, u_h)$ nicht in N_{f_i} (aber in $N_{f_{i+1}}$) enthalten ist, muss $f_{i+1}(e) \neq f_i(e)$ und somit $g_{i+1}(e) \neq 0$ sein
- Da e dann auch nicht in N'_{f_i} ist, muss $e^R = (u_h, u_{h-1})$ in N'_{f_i} sein
- Da N'_{f_i} nur Kanten auf kürzesten Pfaden mit Startknoten s enthält, folgt $d_i(s, u_{h-1}) = d_i(s, u_h) + 1$, was (2) impliziert

Der Algorithmus von Dinitz

Beweis (Fortsetzung)

- Aus $(1 + 2)$ folgt

$$d_i(s, u_\ell) \leq d_i(s, u_{\ell-1}) + 1 \leq \cdots \leq d_i(s, s) + \ell = \ell = d_{i+1}(s, u_\ell)$$

- Somit haben wir folgende Ungleichung bewiesen:

$$\text{Für jeden Knoten } u \in V \text{ gilt } d_i(s, u) \leq d_{i+1}(s, u) \tag{3}$$

- Nun zeigen wir $\delta_i < \delta_{i+1}$ für $i = 1, \ldots, k - 1$
- Sei $P = (u_0, u_1, \ldots, u_{\delta_{i+1}})$ ein kürzester Pfad von $s = u_0$ nach $t = u_{\delta_{i+1}}$ in $N_{f_{i+1}}$ (und somit auch in $N'_{f_{i+1}}$)
- Mit Ungleichung 3 folgt, dass $d_i(s, u_h) \leq d_{i+1}(s, u_h) = h$ für $h = 0, \ldots, \delta_{i+1}$ ist
- Wir unterscheiden nun zwei Fälle:
  - Alle Knoten u_h sind in N'_{f_i} enthalten
  - Mindestens ein Knoten u_h ist nicht in N'_{f_i} enthalten

Der Algorithmus von Dinitz

Beweis (Fortsetzung)

- Alle Knoten u_h sind in N'_{f_i} enthalten
  - In diesem Fall muss ein h mit $d_i(s, u_h) \leq d_i(s, u_{h-1})$ existieren
  - Würde nämlich $d_i(s, u_h) > d_i(s, u_{h-1})$ für $h = 1, \ldots, \delta_{i+1} - 1$ gelten, so wären die Kanten (u_{h-1}, u_h) für $h = 1, \ldots, \delta_{i+1} - 1$ wegen (2) in N_{f_i} enthalten und somit würde wegen (1) $d_i(s, u_h) = d_i(s, u_{h-1}) + 1$ für $h = 1, \ldots, \delta_{i+1} - 1$ folgen
  - Dies hätte wiederum zur Folge, dass P ein kürzester Pfad von s nach t in N_{f_i} und somit ein s-t-Pfad in N'_{f_i} wäre, der von g_i nicht blockiert wird, da er auch in $N_{f_{i+1}}$ existiert
  - Da aber g_i jeden s-t-Pfad in N'_{f_i} blockiert, muss also ein h mit $d_i(s, u_h) \leq d_i(s, u_{h-1})$ existieren und es folgt mit $(1 + 2)$:

$$\delta_i = d_i(s,t) \leq d_i(s,u_h) + \underbrace{d_i(u_h,t)}_{\leq \delta_{i+1} - h} \leq \underbrace{d_i(s,u_{h-1})}_{\leq d_{i+1}(s,u_{h-1}) = h-1} + \delta_{i+1} - h < \delta_{i+1}$$

Der Algorithmus von Dinitz

Beweis (Schluss)

- Mindestens ein Knoten u_h ist nicht in N'_{f_i} enthalten
  - Sei u_h der erste solche Knoten auf P und sei $e = (u_{h-1}, u_h)$
  - Da $u_h \neq t = u_{\delta_{i+1}}$ ist, folgt $d_{i+1}(s, u_h) < d_{i+1}(s, t) = \delta_{i+1}$
  - Zudem liegt die Kante e nicht nur in $N_{f_{i+1}}$, sondern wegen $f_{i+1}(e) = f_i(e)$ (da weder e noch e^R zu N'_{f_i} gehören) auch in N_{f_i}
  - Da somit u_{h-1} in N'_{f_i} und e in N_{f_i} ist, kann u_h nur aus dem Grund nicht zu N'_{f_i} gehören, dass $d_i(s, u_h) = d_i(s, t)$ ist
  - Daher folgt unter Verwendung von $(1 + 2 + 3)$ auch in diesem Fall die Ungleichung $\delta_i < \delta_{i+1}$:

$$\delta_i = d_i(s,t) = d_i(s,u_h) \leq \underbrace{d_i(s,u_{h-1})}_{\leq d_{i+1}(s,u_{h-1})} + 1 \leq d_{i+1}(s,u_h) < \delta_{i+1}$$

□

Korollar

Der Algorithmus von Dinitz berechnet bei Verwendung der Prozedur `blockfluss1` einen maximalen Fluss in Zeit $O(n^2 m)$

Der Algorithmus von Dinitz

- Wir betrachten nun die Prozedur `blockfluss2`
- Zu ihrer Beschreibung benötigen wir folgende Notation

Definition Sei $N = (V, E, s, t, c)$ ein Netzwerk.

- Der ***Durchsatz eines Knotens*** $u \in V$ ***in*** N ist

$$D(u) = \begin{cases} c^+(u), & u = s \\ c^-(u), & u = t \\ \min\{c^+(u), c^-(u)\}, & \text{sonst} \end{cases}$$

  wobei $c^+(u) = \sum_{v \in V} c(u, v)$ die ***Ausgangskapazität*** und $c^-(u) = \sum_{v \in V} c(v, u)$ die ***Eingangskapazität von*** u in N ist

- Ein Fluss g in N ***sättigt einen Knoten*** $u \in V$ in N, falls
  - $u = s$ ist und g alle Kanten $(s, v) \in E$ mit Startknoten s sättigt
  - oder $u = t$ ist und g alle Kanten $(v, t) \in E$ mit Zielknoten t sättigt
  - oder $u \in V - \{s, t\}$ ist und g alle Kanten $(u, v) \in E$ mit Startknoten u oder alle Kanten $(v, u) \in E$ mit Zielknoten u sättigt