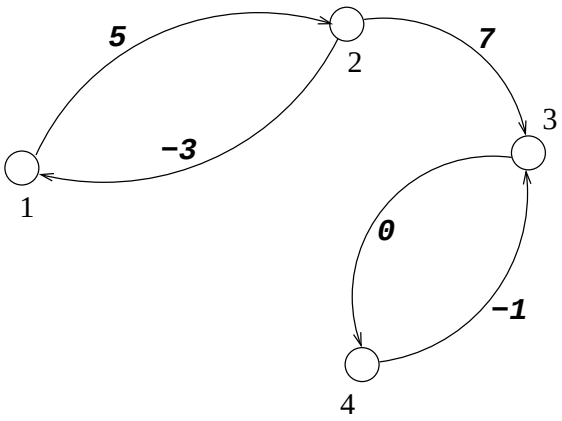
8 Kürzeste Wege

Hier sind alle Graphen gerichtet und gewichtet, d.h. wir haben eine Kostenfunktion $K : E \to \mathbb{R}$ dabei.

Also etwa:



$$K(1,2) = 5, K(2,1) = -3, K(2,3) = -7, K(3,4) = 0$$

Ist $W = (v_0, v_1, \ldots, v_k)$ irgendein Weg im Graphen, so bezeichnet

$$K(W) = K(v_0, v_1) + K(v_1, v_2) + \cdots + K(v_{k-1}, v_k)$$

die Kosten von W. $K(v_0) = 0$

Betrachten wir die Knoten 3 und 4, so ist

$$\begin{aligned} K(3,4) &= 0, \\ K(4,3) &= -1 \quad \text{und} \\ K(3,4,3,4) &= -1, \\ K(3,4,3,4,3,4) &= -2, \\ &\ldots \end{aligned}$$

Negative Kreise erlauben beliebig kurze Wege. Wir beschränken uns auf einfache Wege, d.h. noch einmal, dass alle Kanten verschieden sind.

Definition 8.1(Distanz): *Für $u, v \in V$ ist*

- $\mathrm{Dist}(u,v) = \min\{K(W) \mid W$ *ist einfacher Weg von u nach v* $\}$, *sofern es einen Weg $u \circ\!\!\longrightarrow\!\!\circ\, v$ gibt.*
- $\mathrm{Dist}(u,v) = \infty$, *wenn es keinen Weg $u \circ\!\!\longrightarrow\!\!\circ\, v$ gibt.*
- $Dist(v,v) = 0$

Uns geht es jetzt darum, *kürzeste Wege* zu finden. Dazu machen wir zunächst folgende Beobachtung:

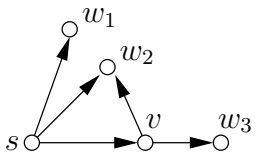
Ist $u = v_0 \to v_1 \to v_2 \cdots v_{k-1} \to v_k = v$ ein kürzester Weg von u nach v, so sind alle Wege $v_0 \to v_i$ oben auch kürzeste Wege. Deshalb ist es sinnvoll, das *single source shortest path-Problem* zu betrachten, also alle kürzesten Wege von einem Ausgangspunkt aus zu suchen. Ist s unser Ausgangspunkt, so bietet es sich an, eine Kante minimaler Kosten von s aus zu betrachten:

$$s \to v$$

Gibt uns diese Kante einen kürzesten Weg von s nach v? Im Allgemeinen nicht! Nur unter der Einschränkung, dass die Kostenfunktion $K : E \to \mathbb{R}^{\geq 0}$ ist, also keine Kosten < 0 erlaubt sind. Diese Bedingung treffen wir zunächst einmal bis auf weiteres.

Also, warum ist der Weg $s \to v$ ein kürzester Weg? (Die nachfolgende Aussage gilt bei negativen Kosten so nicht!) Jeder andere Weg von s aus hat durch seine erste Kante $s \to w$ mindestens die Kosten $K(s, v)$.

Der *Greedy-Ansatz* funktioniert für den ersten Schritt. Wie bekommen wir einen weiteren kürzesten Weg von s aus?

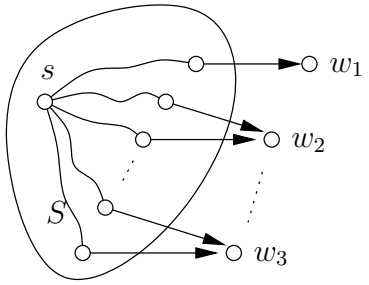


Wir schauen uns die zu s und v adjazenten Knoten an. Oben w_1, w_2, w_3. Wir ermitteln

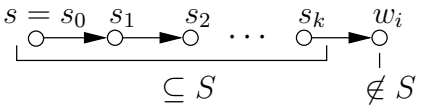
- zu w_1 $K(s, w_1)$
- zu w_2 $\min\{K(s, w_2), K(s, v) + K(v, w_2)\}$
- zu w_3 $K(s, v) + K(v, w_3)$.

Ein minimaler dieser Werte gibt uns einen weiteren kürzesten Weg. Ist etwa $K(s, v) + K(v, w_2)$ minimal, dann gibt es keinen kürzeren Weg $s \to w_2$. Ein solcher Weg müsste ja die Menge $\{s, v\}$ irgendwann verlassen. Dazu müssen aber die eingezeichneten Wege genommen werden und der Weg zu w_2 wird höchstens länger. (Wieder wichtig: Kosten ≥ 0).

So geht es allgemein weiter: Ist S eine Menge von Knoten, zu denen ein kürzester Weg von s aus gefunden ist, so betrachten wir alle Knoten w_i adjazent zu S aber nicht in S.



Für jeden Knoten w_i berechnen wir die minimalen Kosten eines Weges der Art



Für ein w_i werden diese Kosten minimal und wir haben einen kürzesten Weg $s \circ\!\!\longrightarrow\!\!\circ\, w_i$.
Wie vorher sieht man, dass es wirklich keinen kürzeren Weg $s \circ\!\!\longrightarrow\!\!\circ\, w_i$ gibt.

Das Prinzip: Nimm immer den nächstkürzeren Weg von S ausgehend.

8.1 Algorithmus (Dijkstra 1959)

Algorithmus 13: Dijktstras Algorithmus

Input : $G = (V, E),\ K : E \to \mathbb{R}^{\geq 0},\ |V| = n, s \in V$
Output : Array $D[1 \ldots n]$ of real mit $D[v] = \text{Dist}(s, v)$ für $v \in V$
Data : $S \to$ Menge der Knoten, zu denen ein kürzester Weg gefunden ist

```
1  D[s] = 0;
2  S = {s};
3  Q = V \ S;
4  for i = 1 to n − 1 do     /* Wir suchen noch n − 1 kürzeste Wege. */
5  |  foreach w ∈ Q do
   |  |  /* Betrachte alle Kanten (v, w) mit v ∈ S */
6  |  |  D[w] = min{D[v] + K(v, w) | v ∈ S und (v, w) ∈ E};
7  |  end
8  |  w = ein w ∈ Q mit D[w] minimal;
9  |  S = S ∪ {w};
10 |  Q = Q \ {w};
11 end
```

Korrektheit mit der Invariante: Für alle $w \in S_l$ ist $D[w]$ = Kosten eines kürzesten Weges.

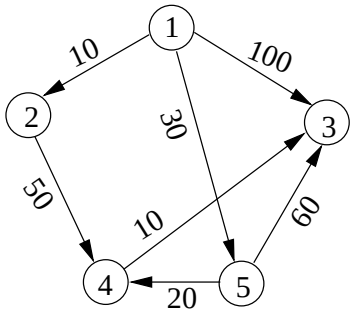
Das Argument gilt wie oben bereits vorgeführt.

Zwecks Merken der Wege das Array $\pi[1,\ldots,n]$ of $\{1,\ldots,n\}$ mit

$$\pi[u] = v \iff \text{Ein kürzester Weg } s \circ\!\!\longrightarrow\!\!\circ\, v \quad \text{ist } (s,\ldots,u,v).$$

π kann leicht im Algorithmus mitgeführt werden. Es ist das Vaterarray des *Kürzesten-Wege-Baumes* mit Wurzel s.

Beispiel 8.1: *Bestimmung der kürzesten Wege von s aus in folgendem Graphen.*

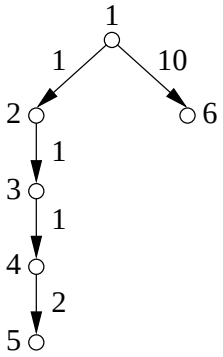


S	w	$D[1]$	$D[2]$	$D[3]$	$D[4]$	$D[5]$
$\{1\}$	/	0	10	100	∞	30
$\{1,2\}$	2	0	10	100	60	30
$\{1,2,5\}$	5	0	10	90	50	30
$\{1,2,5,4\}$	4	0	10	60	50	30
$\{1,2,5,4,3\}$	3	0	10	60	50	30

Die Wege werden der Länge nach gefunden:

Beispiel 8.2:

$G =$



S
$\{1\}$
$\{1,2\}$
$\{1,2,3\}$
$\{1,2,3,4,5\}$
$\{1,2,3,4,5,6\}$

Kürzester-Wege-Baum:

$$\begin{aligned} \pi[1] &= 1, \\ \pi[2] &= 1, \\ \pi[3] &= 2, \\ \pi[4] &= 5, \\ \pi[5] &= 4, \\ \pi[6] &= 1 \end{aligned}$$

Die Laufzeit ist bei direkter Implementierung etwa $O(|V| \cdot |E|)$, da es $O(|V|)$ Schleifendurchläufe gibt und jedesmal die Kanten durchsucht werden, ob sie zwischen S und Q verlaufen. Q und S sind als boolesche Arrays mit

$$Q[v] = \mathit{true} \iff v \in Q$$

zu implementieren.

Die Datenstruktur des Dictionary (Wörterbuch) speichert eine Menge von Elementen und unterstützt die Operationen

- Find(u) = Finden des Elementes u innerhalb der Struktur
- Insert(u) = Einfügen
- Delete(u) = Löschen

Ist die Grundmenge nicht als Indexmenge geeignet, dann Hashing oder Suchbaum verwenden.

In Q ist ein Dictionary implementiert. Jede Operation erfolgt in $O(1)$. Das Ziel ist eine bessere Implementierung.

Für welche $w \in Q$ kann sich $D[w]$ in 6. nur ändern? Nur für diejenigen, die adjazent zu dem w des *vorherigen* Laufes sind. Dann sieht es etwa so aus:

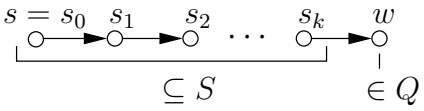
Algorithmus 14: Dijktstras Algorithmus (ohne mehrfache Berechnung derselben $D[w]$)

Input : $G = (V, E),\ K : E \to \mathbb{R}^{\geq 0},\ |V| = n, s \in V$
Output : Array $D[1 \ldots n]$ of real mit $D[v] = \text{Dist}(s, v)$ für $v \in V$
Data : $S \to$ Menge der Knoten, zu denen ein kürzester Weg gefunden ist

```
1  D[s] = 0;
2  D[w] = K(s,w) für w ∈ Adj[s];
3  D[v] = ∞ für v ∈ V \ ({s} ∪ Adj[s]);
4  S = {s};
5  Q = V \ S;
6  for i = 1 to n − 1 do     /* Wir suchen noch n − 1 kürzeste Wege. */
7  |  w = ein w ∈ Q mit D[w] minimal;
8  |  S = S ∪ {w};
9  |  Q = Q \ {w};
10 |  foreach v ∈ Adj[w] do
   |  |  /* D[v] anpassen für v adjazent zu w */
11 |  |  if D[w] + K(w,v) < D[v] then
12 |  |  |  D[v] = D[w] + K(w,v);
13 |  |  end
14 |  end
15 end
```

Korrektheit mit zusätzlicher Invariante:

Für $w \in Q_l$ ist $D_l[w]$ = minimale Kosten eines Weges der Art



Man kann auch leicht zeigen:

> Für alle $v \in S_l$ ist $D_l[v] \leq D_l[w_l]$ mit w_l = das Minimum der l-ten Runde. Damit ändert 12. nichts mehr an $D[v]$ für $v \in S_l$.

Laufzeit:

7.	insgesamt $n-1$-mal Minimum finden
8., 9.	insgesamt $n-1$-mal Minimum löschen
10.-14.	insgesamt $O(\|E\|)$, da dort jede Adjazenzliste nur einmal durchlaufen wird.

Mit Q als boolesches Array:

7.	$O(n^2)$ insgesamt. Das Finden eines neuen Minimums nach dem Löschen dauert $O(n)$.
8., 9.	$O(n)$ insgesamt, einzelnes Löschen dauert $O(1)$.
10.-14.	Bleibt bei $O(\|E\|)$.

Also alles in allem $O(n^2)$.

Aber mit Q benötigen wir die klassischen Operationen der Priority Queue, also Q als Heap, Schlüsselwert aus D.

7.	$O(n)$ insgesamt.
8., 9.	$O(n \cdot \log n)$ insgesamt.
10.-14.	$\|E\|$-mal Heap anpassen, mit DecreaseKey(Q, v, s) (vergleiche Prim, Seite 107) $O(\|E\| \cdot \log n)$. Zum Finden Index mitführen.

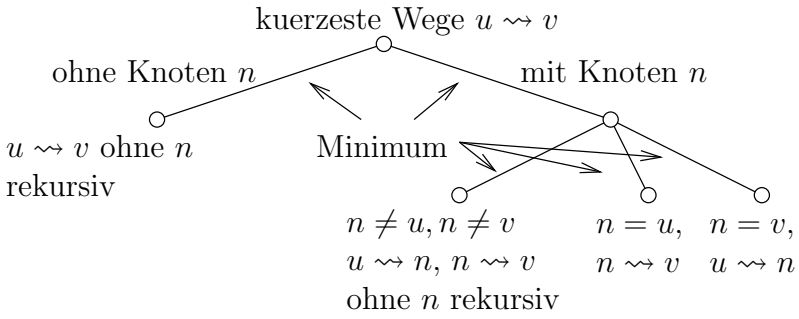
Also:

Q als boolesches Array:	$O(n^2)$ (Zeit fällt beim Finden der Minima an).
Q als heap mit Index:	$O(\|E\| \cdot \log n)$ (Zeit fällt beim DecreaseKey(Q, v, s) an).

Ist $|E| \cdot \log n \geq n^2$, also ist der Graph sehr dicht, dann ist ein Array besser (vergleiche Prim).

8.2 Algorithmus Floyd-Warshall

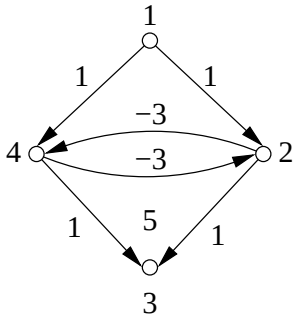
Wir beginnen einmal mit einer anderen Fallunterscheidung beim Backtracking:



Korrektheit: Ist $u, vneqn$ und ist ein kürzester Weg $u \circ\!\!\longrightarrow\!\!\circ v$ ohne n, dann wird dieser nach Induktionsvoraussetzung links gefunden. Enthalte jetzt jeder kürzeste Weg $u \circ\!\!\longrightarrow\!\!\circ v$ den Knoten n. Wird ein solcher unbedingt rechts gefunden?

Nur dann, wenn die kürzesten Wege $u \circ\!\!\longrightarrow\!\!\circ n$ und $n \circ\!\!\longrightarrow\!\!\circ v$ keinen weiteren gemeinsamen Knoten haben. Wenn sie einen gemeinsamen Knoten w haben, so $u \rightsquigarrow w \rightsquigarrow n \rightsquigarrow w \rightsquigarrow v$. Da dieser Weg kürzer ist als jeder Weg ohne n, muss $w \rightsquigarrow n \rightsquigarrow w$ ein Kreis der Länge < 0 sein.

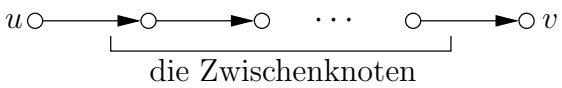
Beispiel 8.3(negative Kreise):

$G =$ 

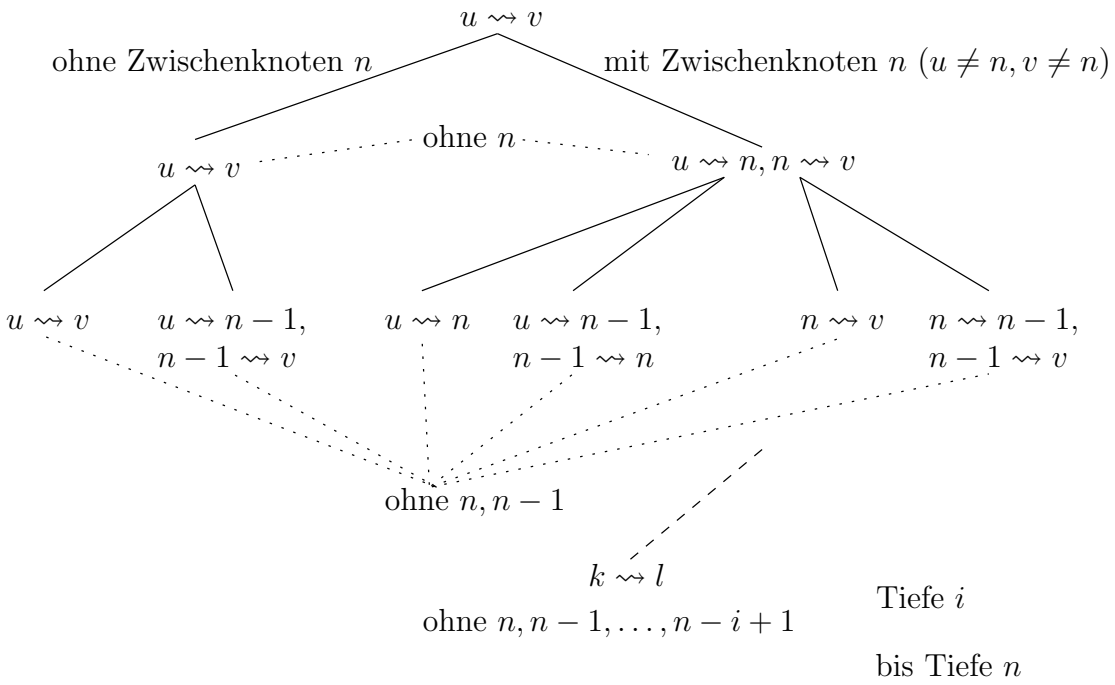
- *Kürzester Weg* $1 \circ\!\!\longrightarrow\!\!\circ 3$ *ohne* 4 *hat Kosten* 2*,*
- *kürzester Weg* $1 \circ\!\!\longrightarrow\!\!\circ 4$ *über* 2 *hat Kosten* -2*,*
- *kürzester Weg* $4 \circ\!\!\longrightarrow\!\!\circ 3$ *ebenfalls über* 2 *hat Kosten* -2
- *und* $(2, 4, 2)$ *ist Kreis der Kosten* -6*.*

Betrachten wir also jetzt $K : E \to \mathbb{R}$, aber so, dass keine Kreise < 0 existieren.

Es ist günstiger, die Fallunterscheidung nach den echten Zwischenknoten zu machen.



Also Backtracking:



Die rekursive Prozedur:

KW(u, v, k) für kürzeste Wege $u \circ\!\!\longrightarrow\!\!\circ\, v$ ohne Zwischenknoten $> k$. Kürzester Weg ergibt sich durch KW(u, v, n).

Rekursionsanfang KW$(u, v, 0)$ gibt Kante $u \circ\!\!\longrightarrow\!\!\circ\, v$. Wieviele verschiedene Aufrufe? $\Rightarrow O(n^3)$

Also dynamisches Programmieren:

$T[u, v, k]$ = kürzester Weg $u \circ\!\!\longrightarrow\!\!\circ\, v$ wobei Zwischenknoten $\subseteq \{1, \ldots, k\}$. (Also nicht unter $k + 1, \ldots, n$)

$$
\begin{aligned}
T[u, v, 0] &= K(u, v) \quad \text{für alle } u, v \\
T[u, v, 1] &= \min\{T[u, 1, 0] + T[1, v, 0], T[u, v, 0]\} \quad \text{für alle } u, v \\
&\vdots \\
T[u, v, n] &= \min\{T[u, n, n-1] + T[n, v, n-1], T[u, v, n-1]\} \quad \text{für alle } u, v
\end{aligned}
$$

Dies ist das *all pairs shortest path*-Problem. Das heißt es werden die kürzesten Wege zwischen *allen* Knoten bestimmt. *Wichtig:* G ohne Kreise mit Länge < 0.

Algorithmus 15: Floyd-Warshall-Algorithmus

Input : $G = (V, E)$ gerichteter Graph, $|V| = n$, $K : E \to \mathbb{R}$
Output : Ein kürzester Weg $u \rightsquigarrow v$ für alle Paare $u, v \in V \times V$

`/* Initialisierung */`

1 $$T[u,v] = \begin{cases} 0 & \text{für } u = v \\ K(u,v) & \text{für } (u,v) \in E \\ \infty & \text{für } u \neq v,\ (u,v) \notin E \end{cases}$$

`/* Wege über Zwischenknoten` $\leq i$ `*/`
2 **for** $k = 1$ **to** n **do**
3 | **foreach** $(u, v) \in V \times V$ **do** `/* alle geordneten Paare */`
4 | | $T[u,v] = \min\{T[u,v],\ T[u,k] + T[k,v]\}$;
5 | **end**
6 **end**

Invariante: Nach $l-$tem Lauf der Schleife enthält $T_l[u, v]$ die Länge eines kürzesten Weges $u \circ\!\!\longrightarrow\!\!\circ\, v$ mit Zwischenknoten $\subseteq \{1, \ldots, l\}$.

Wichtig: Keine negativen Kreise, denn ist in $l + 1$-ter Runde

$$T_l[u, l + 1] + T_l[l + 1, v] < T_l[u, v]$$

dann ist der Weg hinter $T_l[u, l + 1] + T_l[l + 1, v]$ ein einfacher!

Ist $u \rightsquigarrow l{+}1 \rightsquigarrow v$ kürzer als die kürzesten Wege $u \rightsquigarrow v$ ohne $l{+}1$, dann tritt oben die Situation $u \rightsquigarrow w \rightsquigarrow l{+}1 \rightsquigarrow w \rightsquigarrow v$ *nicht* ein, da sonst $K(w \rightsquigarrow l{+}1 \rightsquigarrow w) < 0$ sein müsste.

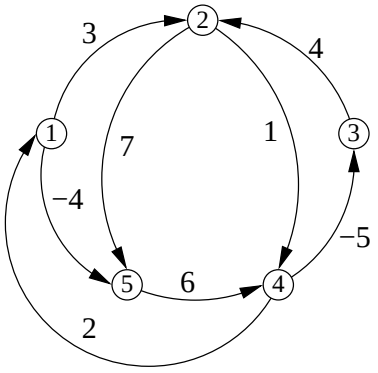
Erkennen von negativen Kreisen: Immer gilt, also bei beliebiger Kostenfunktion, dass Floyd-Warshall die Kosten eines Weges von $u \circ\!\!\longrightarrow\!\!\circ\, v$ in $T[u, v]$ liefert. Dann ist

$$T[u, u] < 0 \iff G \text{ hat Kreis } < 0.$$

Laufzeit: $O(n^3)$ (Vergleiche Dijkstra $O(|E| \log |V|)$ oder $O(|V|^2)$.)

Beispiel 8.4: *Die ersten Schritte des Algorithmus auf dem folgenden Graphen.*

$G =$



Am Anfang:

	1	2	3	4	5
1	0	3	∞	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

Bevor k *auf 2 geht:*

	1	2	3	4	5
1	0	3	∞	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	**5**	-5	0	∞
5	∞	∞	∞	6	0

Es ist $A[i,j] = \min\{A[i,j], A[i,\mathbf{1}] + A[\mathbf{1},j]\}$

Direkte Wege, d.h. hier Wege mit Zwischenknoten 1.

Bevor k auf 3 geht: Direkte Wege + Wege mit Zwischenknoten 1 *+ Wege mit Zwischenknoten* $1, 2$. Nicht: *mit 2 Zwischenknoten!*

9 Flüsse in Netzwerken

$G = (V, E)$ gerichtet, Kostenfunktion $K : E \to \mathbb{R}^{\geq 0}$. Hier nun $K(u, v) =$ die Kapazität von (u, v). Wir stellen uns vor, (u, v) stellt eine Verbindung dar, durch die etwas fließt (Wasser, Strom, Fahrzeuge, ...).

Dann besagt $K(u, v) = 20$ zum Beispiel

- ≤ 20 Liter Wasser pro Sekunde
- ≤ 10 produzierte Waren pro Tag
- ...

Definition 9.1(Flussnetzwerk): *Ein Flussnetzwerk ist ein gerichteter Graph* $G = (V, E)$ *mit* $K : V \times V \to \mathbb{R}^{\geq 0}$, $K(u, v) = 0$ *falls* $(u, v) \notin E$.
Außerdem gibt es zwei ausgezeichnete Knoten

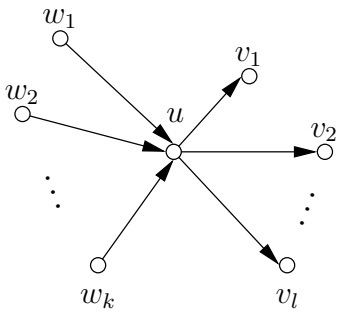
- $s \in V$, *Quelle (source)*
- $t \in V$, *Ziel (target, sink)*

Wir verlangen außerdem noch: Jeder Knoten $v \in V$ *ist auf einem Weg* $s \rightsquigarrow v \rightsquigarrow t$.

Ziel: Ein möglichst starker Fluss pro Zeiteinheit von s nach t.

Den Fluss durch $u \circ\!\!\longrightarrow\!\!\circ\, v \in E$ bezeichnen wir mit $f(u, v) \in \mathbb{R}^{\geq 0}$. Für einen Fluss f müssen die folgenden Bedingungen gelten:

- $f(u, v) \leq K(u, v)$
- Was zu u hinfließt, muss auch wieder wegfließen (sofern $u \neq s, u \neq t$). Also

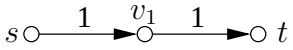


dann $\sum f(w_i, u) = \sum (u, v_i)$.

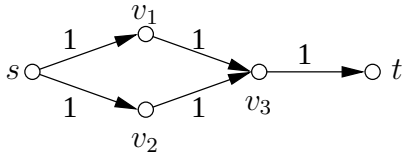
Prinzip: Methode der Erweiterungspfade (Ford - Fulkerson 1950er Jahre)

1. Beginne mit dem Fluss 0, $f(u,v) = 0$ für alle (u,v)
2. Suche einen Weg (Erweiterungspfad) $(s = v_0 \to v_1 \to v_2 \dots v_{k-1} \to v_k = t)$ so dass für alle $f(v_i, v_{i+1}) < K(v_i, v_{i+1})$.
3. Erhöhe den Fluss entlang des Weges so weit es geht. Dann bei 2. weiter.

Dieses Problem ist von ganz anderem Charakter als bisher, da die Anzahl der zunächst zulässigen Flüsse unendlich ist. Wir betrachten folgendes Flussnetzwerk:

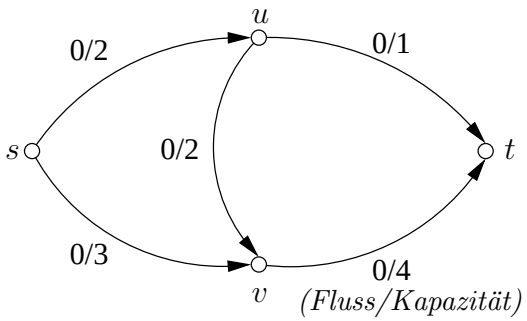


Ist $f(s,v_1) = f(v_1,t) < 1$ so haben wir einen erlaubten Fluss. Aber sogar maximale Flüsse gibt es unendlich viele:

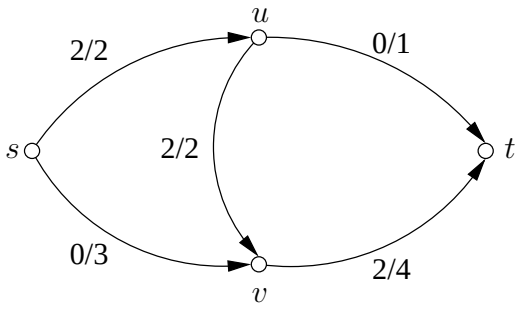


Jeder Fluss mit $f(s,v_1) + f(s,v_2) = 1$ ist maximal.

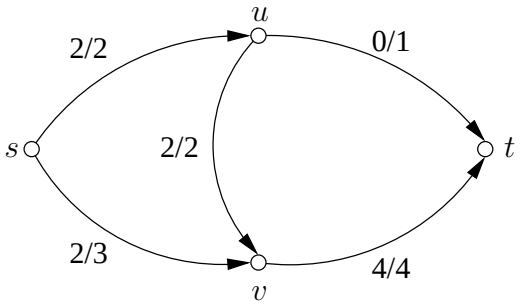
Beispiel 9.1(negative Flüsse): *Wir betrachten das folgende Flussnetzwerk:*



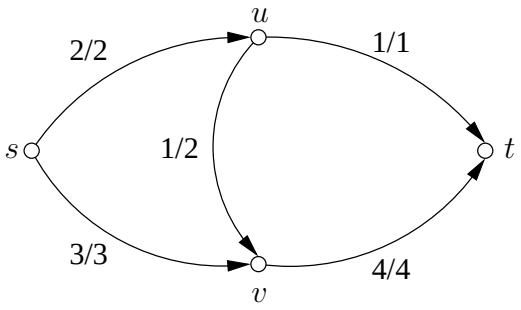
Weg (s,u,v,t) $+2$



Weg (s, v, t) $+2$

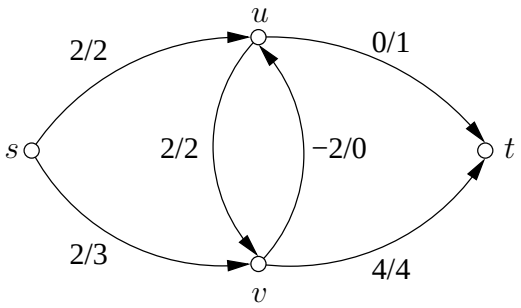


Jetzt gibt es keinen Erweiterungspfad mehr, aber der Fluss in



ist größer!

Mit negativen Flüssen: Immer ist $f(u, v) = -f(v, u)$.



Weg (s, v, u, t) $+1$ *gibt den maximalen Fluss. Wir lassen auch* $f(u, v) < 0$ *zu.*

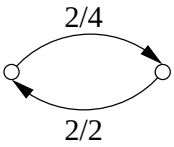
Definition 9.2(Fluss): *Ein Fluss ist eine Funktion* $f : V \times V \to \mathbb{R}$ *mit*

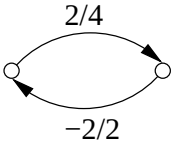
- $f(u, v) \leq K(u, v)$ *für alle* u, v *(Kapazitätsbedingung)*
- $f(u, v) = -f(v, u)$ *für alle* u, v *(Symmetrie)*
- *Für alle* $u \neq s,\ u \neq t$ *gilt* $\sum_{v \in V} f(u, v) = 0$ *(Kirchhoffsches Gesetz)*

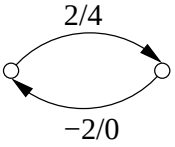
- $|f| = \sum_{v \in V} f(s, v)$ *ist der Wert von* f.

Problem: Wie groß ist der maximale Fluss $|f|$ in einem gegebenen Flussnetzwerk?

Noch einige Anmerkungen:

- Immer ist $f(u, u) = -f(u, u) = 0$, da $K(u, u) = 0$, da nie $(u, u) \in E$.
- Auch $f(u, v) = f(v, u) = 0$, wenn $(u, v), (v, u) \notin E$.
  Denn es ist $f(u, v) = -f(v, u)$, also einer der Werte > 0.
- Ist $f(u, v) \neq 0$, so $(u, v) \in E$ oder $(v, u) \in E$.
- Nicht sein kann 

  wegen $f(u, v) = -f(v, u)$.
  Hier würden wir setzen $f(u, v) = f(v, u) = 0$. Wir können (müssen) kürzen, aber 

  oder 

  geht.

Definition 9.3(Restnetzwerk): *Gegeben ist das Flussnetzwerk* $G = (V, E)$ *mit Kapazität* $K : V \times V \to \mathbb{R}^{\geq 0}$ *und ein zulässiger Fluss* $f : V \times V \to \mathbb{R}$.

- *Das Restnetzwerk* G_f *mit Restkapazität* K_f *ist gegeben durch:*

$$\begin{aligned} K_f(u, v) &= K(u, v) - f(u, v) \geq 0 \\ E_f &= \{(u, v) \mid K_f(u, v) > 0\} \\ G_f &= (V, E_f) \end{aligned}$$

  Es ist $K_f(u, v) \geq 0$, *da* $f(u, v) \leq K(u, v)$.

- *Ein Erweiterungspfad von* G *und* f *ist ein einfacher Weg in* G_f

$$W = (s = v_0 \to v_1 \to v_2 \ldots v_{k-1} \to v_k = t)$$

  Die Restkapazität von W *ist*

$$K_f(W) = \min\{K_f(v_i, v_{i+1})\},$$

  nach Definition gilt $K_f(v_i, v_{i+1}) \gneq 0$).

Es ist G_f mit der Restkapazität K_f ein Flussnetzwerk. *Beachte:* $K_f(u, v)$ ist für alle Paare $(u, v) \in V \times V$ definiert.

Wir betrachten jetzt $g : V \times V \to \mathbb{R}$ mit

$$\begin{aligned} g(v_i, v_{i+1}) &= K_f(W) > 0 \\ g(v_{i+1}, v_i) &= -K_f(W) = -g(v_i, v_{i+1}) \\ g(u, v) &= 0 \quad \text{für} (u, v) \notin W. \end{aligned}$$

Das ist ein zulässiger Fluss auf G_f. Das heißt: $|g| = K_f(W)$.

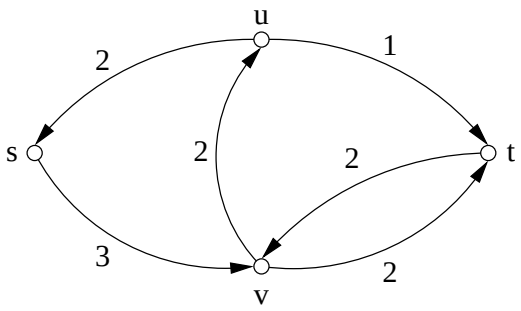
Tatsächlich gilt nun sogar folgendes:

Lemma 9.1: *Sei G, K, f Flussnetzwerk mit zulässigem Fluss f. Und sei G_f das zugehörige Restnetzwerk.*

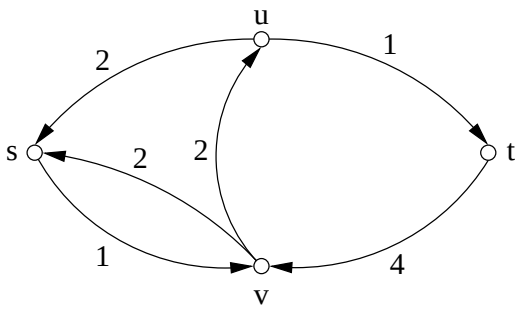
Sei jetzt g irgendein zulässiger Fluss auf G_f. Dann ist $f + g$ ein gültiger Fluss auf G und $|f + g| = |f| + |g|$.

Beispiel 9.2(Restnetzwerke): *Zunächst noch ein Beispiel. Wir betrachten die Restnetzwerke zu dem Flussnetzwerk aus Beispiel 9.1.*

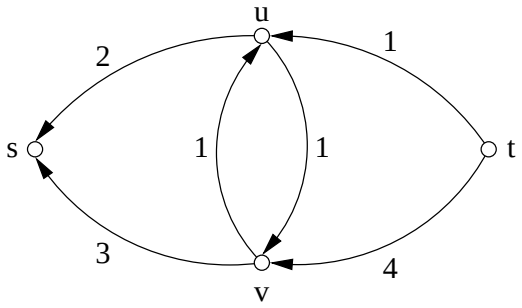
Weg (s, u, v, t):



Weg (s, v, t):



Weg (s, v, u, t)*:*



kein Erweiterungspfad. Kapazitäten sind immer ≥ 0*.*

Beweis. Wir müssen also überlegen, dass $f + g$ ein zulässiger Fluss in G ist.

- Kapazitätsbedingung: Es ist $g(u, v) \leq K_f(u, v) = K(u, v) - f(u, v)$. Also

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

- Symmetrie:

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

- Kirchhoff: Sei $u \in V \setminus \{s, t\}$, dann

$$\begin{aligned}\sum_{v\in V}(f+g)(u,v) &= \sum_{v\in V}\big(f(u,v) + g(u,v)\big) \\ &= \sum_{v\in V} f(u,v) + \sum_{u\in V} g(u,v) \\ &= 0.\end{aligned}$$

Schließlich ist $|f + g| = \sum_{v\in V}(f + g)(s, v) = |f| + |g|$. □

9.1 Algorithmus nach Ford-Fulkerson

Algorithmus 16: Maximaler Fluss (Ford-Fulkerson)

```
   /* Starte beim Nullfluss */
1  foreach (u, v) ∈ V × V do
2  |  f(u, v) = 0;
3  end
4  while Es gibt einen Weg s ⟶ t  in G_f do
5  |  W = ein Erweiterungspfad in G_f;
6  |  g = Fluss in G_f mit g(u, v) = K_f(W) für alle (u, v) ∈ W;
7  |  f = f + g;
8  end
   /* Gib f als maximalen Fluss aus */
9  return f;
```

Warum liefert dieses Vorgehen einen *maximalen Fluss*? Dass auf diese Weise ein korrekter Fluss erzeugt wird, ist klar. Aber ist dieser auch *maximal*? Dazu müssen wir etwas weiter ausholen. Zunächst noch eine Definition.

Definition 9.4(Schnitt): *Ein Schnitt eines Flussnetzwerkes $G = (V, E)$ ist eine Partition S, T von V, d.h. $V = S \cup T$ mit $s \in S$, $t \in T$ und $S \cap T = \emptyset$.*

- *Kapazität von S, T*

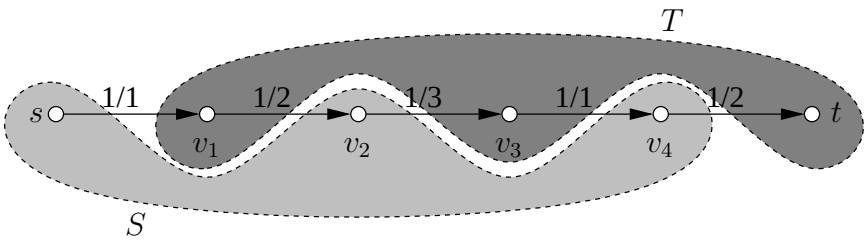
$$K(S,T) = \sum_{u \in S, v \in T} K(u,v) \qquad (K(u,v) \geq 0, u \in S, v \in T)$$

- *Ist f ein Fluss, so ist der Fluss durch den Schnitt*

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v).$$

Immer ist $f(S,T) \leq K(S,T)$ und $|f| = f\big(\{s\}, V \setminus \{s\}\big)$.

Beispiel 9.3(Schnitt): *Wir betrachten ein Flussnetzwerk:*



$$\begin{aligned} S &= \{s, v_2, v_4\} \\ T &= \{v_1, v_3, t\} \end{aligned}$$

$$\begin{aligned} K(S,T) &= \underbrace{K(s,v_1)}_{=1} + \underbrace{K(v_2,v_3)}_{=3} + \underbrace{K(v_4,t)}_{=2} = 6 \\ F(S,T) &= 1-1+1-1+1=1 \quad \textit{(Betrag des Flusses)} \end{aligned}$$

S, T in mehreren Stücken $\longrightarrow$ kein minimaler Schnitt.

Tatsächlich gilt sogar für jeden Schnitt S, T, dass $f(S,T) = |f|$ ist.

Beweis. Induktion über $|S|$.

- Induktionsanfang: $|S| = 1$, dann $S = \{s\}$, und $f(S,T) = |f|$ nach Definition.
- Induktionsschritt: Sei $|S| = l + 1$, $v \in S$, $v \neq s$. Wir betrachten den Zustand aus dem dieses S entstanden ist:

  $$\begin{aligned} S' &= S \setminus \{v\}, \quad |S'| = l \quad \text{(Behauptung gilt nach Induktionsvoraussetzung!)} \\ T' &= T \cup \{v\} = V \setminus S'. \end{aligned}$$

  Dann ist der neue Fluss durch den Schnitt

  $$f(S,T) = f(S',T') - \underbrace{\sum_{u \in S} f(u,v)}_{=-\sum_{u\in S} f(v,u)} + \sum_{u \in T} f(v,u) = |f| + \underbrace{\sum_{u \in V} f(v,u)}_{=0} = |f|.$$

Also: Für jeden Schnitt $|f| = f(S,T) \leq K(S,T)$. □

Aus dieder Überlegung folgt auch, dass

$$\max\{|f| \mid f \text{ Fluss}\} \leq \min\{K(S,T) \mid S,T \text{ Schnitt}\}.$$

Wir zeigen im folgenden: Es gibt einen Fluss f^*, der diese obere Schranke erreicht, das heißt

$$f^* = \min\{K(S,T) \mid S,T \text{ Schnitt}\}.$$

Satz 9.1(Min-Cut-Max-Flow): *Ist f ein zulässiger Fluss in G, so sind die folgenden Aussagen äquivalent.*

1. *f ist ein maximaler Fluss.*
2. *G_f hat keinen Erweiterungspfad.*
3. *Es gibt einen Schnitt S, T, so dass $|f| = K(S,T)$.*

(Immer $f(S,T) = |f|$, $K(S,T) \geq |f|$)

Beweis.

- *(1)* $\rightarrow$ *(2)* gilt, da f maximal. Gälte *(2)* nicht, dann gälte auch *(1)* nicht.
- *(2)* $\rightarrow$ *(3)*: Setze

$$\begin{aligned} S &= \{u \in V \mid \text{Es gibt Weg } (s,u) \text{ in } G_f\} \\ T &= V \setminus S \end{aligned}$$

  Dann $s \in S$ und $t \in T$, da kein Erweiterungspfad nach *(2)*. Also ist S, T ein ordentlicher Schnitt.

  Für $u \in S$, $v \in T$ gilt:

$$0 = K_f(u,v) = K(u,v) - f(u,v)$$

  Also $K(u,v) = f(u,v)$. Dann

$$\begin{aligned} |f| = f(S,T) &= \sum_{u \in S, v \in T} f(u,v) \\ &= \sum_{u \in S, v \in T} K(u,v) \\ &= K(S,T). \end{aligned}$$

- *(3)* $\rightarrow$ *(1)* gilt, da immer $|f| \leq K(S,T)$.

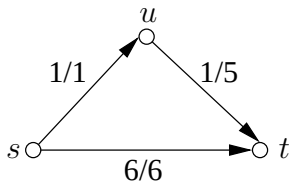
□

Korrektheit von Ford-Fulkerson. Invariante: f_l ist zulässiger Fluss

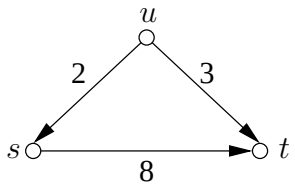
Quintessenz: Am Ende ist f_l maximaler Fluss (Min-Cut-Max-Flow).

Termination: Bei ganzzahligen Kapazitäten terminiert das Verfahren auf jeden Fall. Der Fluss wird in jedem Schritt um eine ganze Zahl ≥ 1 erhöht. Also ist irgendwann nach *endlich* vielen Schritten der maximale Fluss erreicht.

Beispiel 9.4: *Minimaler Schnitt = Maximaler Fluss*



$$\begin{aligned} S &= \{s\},\ T = \{u,t\} \\ K(S,T) &= f(S,T) = 7 \end{aligned}$$



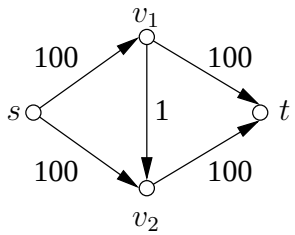
$$\begin{aligned} S = \{s\},\ T = \{u,t\},\ K(S,T) &= 8 \\ S = \{s,u\},\ T = \{t\},\ K(S,T) &= 11 \end{aligned}$$

$$|f| \leq 8.$$

Vergleiche hierzu Beispiel 9.3 auf Seite 125.

Laufzeit von Ford Fulkerson? Bei ganzzahligen Kapazitäten reicht $O(|E|\cdot|f^*|)$, wobei f^* ein maximaler Fluss ist. Bei rationalen Zahlen: Normieren.

Beispiel 9.5(lange Laufzeit bei Ford-Fulkerson): *Wir betrachten das folgende Flussnetzwerk:*



Der maximale Fluss in diesen Netzwerk beträgt offensichtlich $|f| = 200$*. Wenn die Erweiterungswege „schlecht“ gewählt werden, kann sich folgendes ergeben:*

$$\begin{aligned}(s, v_1, v_2, t) &\rightarrow f + 1\\ (s, v_2, v_1, t) &\rightarrow f + 1\\ (s, v_1, v_2, t) &\rightarrow f + 1\\ &\vdots\\ (s, v_2, v_1, t) &\rightarrow f + 1\end{aligned}$$

Also 200 *einzelne Erhöhungen. Beachte im Restnetzwerk das „oszillieren“ von* (v_1, v_2)*.*

Man sieht, dass Laufzeiten der Größe $\Omega(|E| \cdot |f^*|)$ durchaus auftreten können. Die Kapazitäten lassen sich so wählen, dass $|f^*|$ exponentiell in der Länge (in Bits) von G und der Kapazitätsfunktion ist.

Abhilfe schafft hier das folgende vorgehen:

> Wähle im Restnetzwerk immer den kürzesten Erweiterungspfad. Das heißt den Erweiterungspfad mit der minimalen Kantenanzahl nehmen.

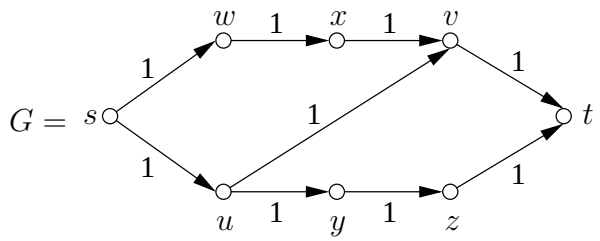
9.2 Algorithmus nach Edmonds-Karp

Algorithmus 17: Maximaler Fluss (Edmonds-Karp)

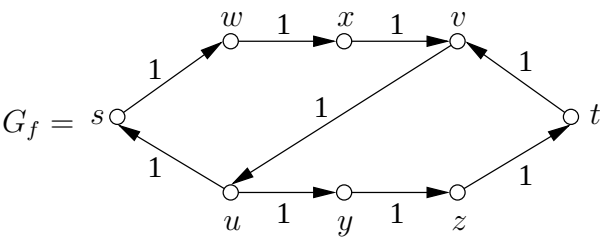
```
1  foreach (u, v) ∈ V × V do
2  |  f(u, v) = 0;
3  end
4  G_f = G;
5  while t in G_f von s aus erreichbar do
6  |  BFS(G_f, s);                                   /* Breitensuche */
7  |  v_k = t;
8  |  v_{k-1} = π[v_k];                             /* π Breitensuchbaum */
9  |  ...
10 |  v_1 = π[v_2];
11 |  s = v_0 = π[v_1];
   |  /* W ist ein kürzester Erweiterungspfad bezogen auf die
   |     Anzahl der Kanten. */
12 |  W = (s = v_0, v_1, ..., v_k = t);
13 |  g(v_i, v_{i+1}) = K_f(W);      /* Flusserhöhung für diesen E-Pfad. */
14 |  f = f + g;
15 |  G_f berechnen;
16 end
   /* Ausgabe: maximaler Fluss f */
17 return f;
```

Beachte: Gemäß Algorithmus ist ein Erweiterungsweg ein Weg ($s \rightsquigarrow t$) mit minimaler Kantenanzahl. Vergleiche das Beispiel 9.5, dort wird gerade kein solcher Weg gewählt.

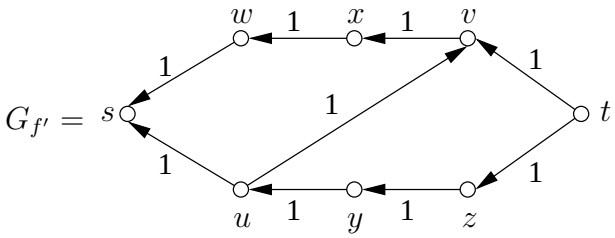
Beispiel 9.6: *Auch bei Edmonds-Karp sind die Umkehrkanten weiterhin notwendig. (Negativer Fluss $\Longleftrightarrow$ Fluss umlenken.)*



Kürzester Erweiterungsweg:
$(s, u, v, t) \ \rightarrow +1$

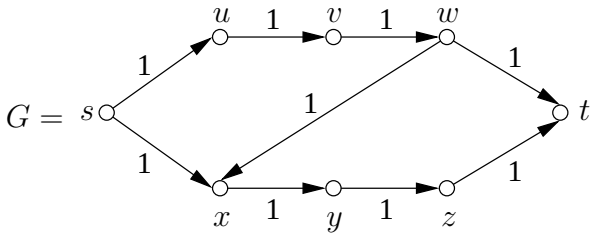


Im Restnetzwerk kürzester Erweiterungsweg:
$(s, w, x, v, u, y, z, t) \ \rightarrow +1$



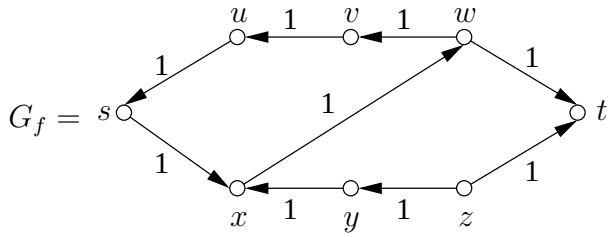
Kein weiterer Erweiterungsweg → maximaler Fluss gefunden.

Beispiel 9.7: *Wenn* nicht *die kürzesten Wege als Erweiterungspfade gewählt werden, können sich die Wege im Restnetzwerk verkürzen.*



$$\begin{aligned} \text{Dist}(s,w) &= 3 \\ \text{Dist}(s,t) &= 4 \end{aligned}$$

Erweiterungsweg
$(s,u,v,w,x,y,z,t) \to +1$



$$\begin{aligned} \text{Dist}(s,w) &= 2 \\ \text{Dist}(s,t) &= 3 \end{aligned}$$

Erweiterungsweg
$(s,x,w,t) \to +1$

Wir zeigen im restlichen Verlauf dieses Abschnitts den folgenden Satz.

Satz 9.2: *Sind* $W_1, W_2, W_3, \ldots W_k$ *die von der* Edmonds-Karp-*Strategie betrachteten Erweiterungswege, dann gilt:*

(a) $\text{Länge}(W_1) \leq \text{Länge}(W_2) \leq \cdots \leq \text{Länge}(W_k) \leq |V|$, *(*$|V|$ = *#Knoten des betrachteten Flussnetzes)*

(b) $\text{Länge}(W_1) \lneq \text{Länge}(W_{|E|+1}) \lneq \text{Länge}(W_{2|E|+1}) \lneq \text{Länge}(W_{3|E|+1}) \lneq \ldots$
Das heißt: Es gibt maximal $|E|$ *viele Erweiterungswege von gleicher Länge.*

Aus diesem Satz folgt dann:

Folgerung 9.1: *Bei Verwendung der* Edmonds-Karp-*Strategie gilt:*

(a) Es werden maximal $|E| \cdot |V|$ Runden der Schleife (5-15) ausgeführt.

(b) Die Laufzeit beträgt $O(|E|^2 \cdot |V|)$, sofern die Kapazitäten in $O(1)$ zu bearbeiten sind.

Wir zeigen zunächst, dass die Folgerung gilt. Danach kommt noch der Beweis des obigen Satzes (der Voraussetzung).

Beweis. (a) Mit der Aussage (b) im Satz gilt für die Wege folgendes:

$$\underbrace{W_1, W_2, \ldots,}_{\text{Länge}\geq 1} \underbrace{W_{|E|+1}, W_{|E|+2}, \ldots,}_{\text{Länge}\geq 2} \underbrace{W_{2|E|+1}, W_{|E|+2}, \ldots}_{\text{Länge}\geq 3} \cdots$$

Also ist sicher beim Index $(|V|-1) \cdot |E| + 1$ die Länge $|V|$ erreicht. Wir haben maximal $|E|$ Wege der Länge $|V|$ und bei $|E| \cdot |V|$ ist ganz sicher der letzte Weg erreicht. Damit hat die Schleife maximal $|E| \cdot |V|$ viele Durchläufe.

(b) Ein Durchlauf durch die Schleife braucht eine Zeit von $O(|V| + |E|)$ für die Breitensuche, Berechnung des Restnetzes etc. Unter der vernünftigen Annahme $|E| \geq |V|$ gilt die Behauptung.

□

Betrachten wir nun einmal einen einzelnen Lauf der Schleife. Zunächst vom allgemeinen *Ford-Fulkerson*. Wir benutzen die folgenden Bezeichnungen:

$$\begin{aligned} f &= \text{Fluss vor dem Lauf der Schleife,} \\ f' &= \text{Fluss nach dem Lauf der Schleife,} \\ G_f, G_{f'} &\quad \text{sind die zugehörigen Restnetzwerke.} \end{aligned}$$

Es gilt: $G_{f'}$ entsteht aus G_f, indem eine (oder mehrehre) Kanten $u \circ\!\!\longrightarrow\!\!\circ\, v$ gelöscht und durch $v \circ\!\!\longrightarrow\!\!\circ\, u$ ersetzt werden. Dabei gilt für $u \circ\!\!\longrightarrow\!\!\circ\, v$:

- Die Kanten $u \circ\!\!\longrightarrow\!\!\circ\, v$ liegen auf dem gewählten Erweiterungsweg.
- Es ist möglich, dass eine Umlenkung $v \circ\!\!\longrightarrow\!\!\circ\, u$ bereits in G_f vorhanden ist. (Das macht nichts.)

Bei *Edmonds-Karp* gilt zusätzlich:

- Die Kanten $u \circ\!\!\longrightarrow\!\!\circ\, v$ wie oben liegen auf einem *kürzesten* Weg $s \circ\!\!\longrightarrow\!\!\circ\, t$.

Wir betrachetn einmal alles ganz allgemein.

Lemma 9.2: *Sei G ein gerichteter Graph mit Knoten s und t. Sei $k = \mathrm{Dist}(s,t)$ die Anzahl der Kanten eines kürzesten Weges von s nach t.*

Ist $u \circ\!\!\longrightarrow\!\!\circ\, v$ eine Kante auf irgendeinem kürzesten Weg $s \circ\!\!\longrightarrow\!\!\circ\, t$, so sei auch die Umkehrkante $v \circ\!\!\longrightarrow\!\!\circ\, u$ in G enthalten.

(a) Ist $u \circ\!\!\longrightarrow\!\!\circ\, v$ Kante eines kürzesten Weges, so kommt nie $v \circ\!\!\longrightarrow\!\!\circ\, u$ auf einem kürzesten Weg vor.

(b) Enthält ein einfacher *kürzester Weg W der Länge k mit l Umkehrkanten, dann gilt für die Weglänge*

$$\text{Länge}(W) \geq k + 2l.$$

Beweis. (a) Angenommen auf den kürestem Wegen kommen irgendwelche dieser Umkehrkanten vor. Wir betrachten einen kürzesten Weg auf dem eine dieser Umkehrkanten am weitesten vorne steht. Dieser sei

$$W = (s = v_0, v_1, \ldots, v_k = t)$$

und (v_i, v_{i+1}) sei die erste Umkehrkante. Außerdem benutzt kein kürzester Weg auf seinen ersten i Schritten eine Umkehrkante.

Angenommen, es gibt einen kürzesten Weg U der Art

$$U = (s = u_0, \ldots, u_j, u_{j+1}, \ldots, u_m = t),$$

mit $u_{j+1} = v_i$ und $u_j = v_{i+1}$. Dann ist $j \geq i$, da sonst die Umkehrkante auf U unter den ersten i Schritten wäre.

Dann ist aber

$$U' = (\underbrace{s = v_0, v_1, \ldots, v_i}_{\text{von } W} = \underbrace{u_{j+1}, \ldots, u_k = t}_{\text{von } U})$$

ein Weg mit

$$\text{Länge}(U') = i + k - (j+1) \leq k - 1,$$

da $j \geq i$. Das kann aber nicht sein, da wir von $\mathrm{Dist}(s,t) = k$ ausgegangen sind!

(b) Zunächst einige Spezialfälle:

- $l = 0$: ✓, da $\text{Dist}(s,t) = k$.
- $l = 1$: Wir betrachten den Weg U.

$$U = (\underbrace{s = u_0, u_1, \ldots, u_i}_{=U_1}, \underbrace{u_{i+1}, \ldots, u_m = t}_{=U_2})$$

Wobei (u_i, u_{i+1}) die Umkehrkante aus einem kürzesten Weg W ist.

$$W = (s = v_0, v_1, \ldots, v_j, v_{j+1}, \ldots, v_k = t), \quad v_j = u_{i+1},\ v_{j+1} = u_i$$

Dann ist

$$\text{Länge}(U_1) = i \geq j + 1,$$

da W ein kürzester Weg ist. Ebenso ist

$$\text{Länge}(U_2) = m - i \geq k - j.$$

Dann ist

$$\text{Länge}(U) \geq (j+1) + 1 + (k-j) = k + 2.$$

- Für ein beliebiges $l \geq 2$ gehen wir analog zum Fall $l = 1$ induktiv vor. Gelte die Behauptung für alle Wege mit $l-1$ Umkehrkanten. Wir betrachten jetzt einen Weg U mit *genau* l Umkehrkanten.

$$U = (\underbrace{s = u_0, u_1, \ldots, u_i}_{U_1}, \underbrace{u_{i+1}, \ldots, u_m = t}_{U_2})$$

Sei (u_i, u_{i+1}) die erste Umlenkung auf U und W ein kürzester Weg.

$$W = (\underbrace{s = v_0, v_1, \ldots, v_j}_{=W_1}, \underbrace{v_{j+1}, \ldots, v_k = t}_{=W_2}), \quad v_j = u_{i+1},\ v_{j+1} = u_i$$

Wir sehen uns jetzt wieder einen Weg U' ohne die Umkehrung an.

$$U' = (\overbrace{\underbrace{s = v_0, \ldots, v_j}_{\substack{\text{ohne Umkehrung} \\ \text{wegen (a)}}}}^{W_1 \text{ von } W} = \overbrace{\underbrace{u_{i+1}, \ldots, u_m = t}_{l-1 \text{ Umkehrungen}}}^{U_2 \text{ von } U})$$

Für die Weglänge ergibt sich nach Induktionsvoraussetzung

$$\text{Länge}(U') \geq k + 2(l-1),$$

da W ein kürzester Weg ist.

$$\text{Länge}(U_1) = i \geq j + 1$$

Damit ist schließlich

$$\begin{aligned} \text{Länge}(U) &= \text{Länge}(U') - \underbrace{\text{Länge}(W_1)}_{=j} + \underbrace{\text{Länge}(U_1)}_{=i} + 1 \\ &\geq k + 2(l-1) - j + \underbrace{(j+1)}_{\geq i} + 1 \\ &= k + 2l. \end{aligned}$$

□

Aus diesem Lemma folgt nun direkt der Satz 9.2.

Beweis. In jedem Lauf der Schleife löscht Edmonds-Karp mindestens eine Kante auf einem kürzesten Weg $s \circ\!\!\longrightarrow\!\!\circ\, t$ und trägt die Umkehrkante ein.

(a) Fangen wir mit W_1 mit Länge$(W_1) = k$ an. Zunächst bearbeitet Edmonds-Karp alle Erweiterungswege der Länge k. Solange es solche Wege gibt, wird nach dem Lemma keine neue Unkehrkante betreten. Irgendwann sind die Wege der Länge k erschöpft. Dann haben wir nur noch Kürzeste Wege der Länge $k' \geq k+1$. Wir bearbeiten diese Wege und machen so weiter.

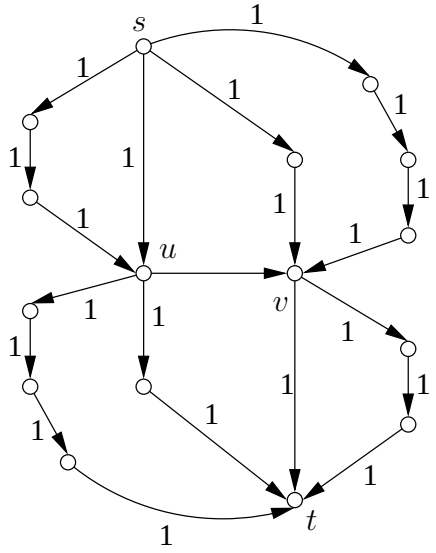
Prinzip: Umkehrkanten von kürzesten Wegen bringen *keine kürzeren* Wege.

(b) Seien einmal $W_1, \ldots, W_l$ die kürzesten Wege in einem Restnetzwerk. In jeder Runde nach Edmonds-Karp verschwindet mindestens eine Kante auf den Kürzesten Wegen.

Nach $|E|$ Runden sind alle Kanten auf den $W_1, \ldots, W_l$ sicherlich fort. Nach dem Lemma enthalten die W_i untereinander keine Umkehrkanten. Also muss in Runde $|E|+1$ eine Umkehrkante vorkommen. Dann vergrößert sich die Länge.

□

Beispiel 9.8(Oszilation bei Edmonds-Karp): *Wir betrachten das folgende Flussnetzwerk.*



1. *Erweiterungsweg* $(s, u, v, t) \to +1$
2. *Erweiterungsweg* $(s, \cdot, v, u, \cdot, t) \to +1$
3. *Erweiterungsweg* $(s, \cdot, \cdot, u, v, \cdot, \cdot, t) \to +1$
4. *Erweiterungsweg* $(s, \cdot, \cdot, \cdot, v, u, \cdot, \cdot, \cdot, t) \to +1$

Die Wege mit der oszillierenden Kante werden immer länger.