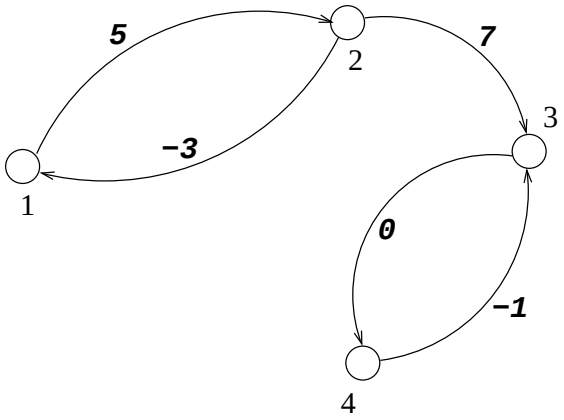
8 Kürzeste Wege

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

Also etwa:

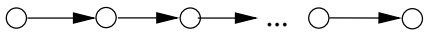


$K(1,2) = 5, K(2,1) = -3, K(2,3) = -7, K(3,4) = 0$

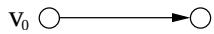
Ist $W = (v_0, v_1, \ldots, v_k)$ irgendein Weg im Graphen, so bezeichnet $K(W) = K(v_0, v_1) + K(v_1, v_2) + \cdots + K(v_{k-1}, v_k)$ die Kosten von W. $K(v_0) = 0$
Betrachten wir die Knoten 3 und 4, so ist $K(3,4) = 0, K(4,3) = -1$ und $K(3,4,3,4) = -1, K(3,4,3,4,3,4) = -2, \ldots$
Negative Kreise 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 $Dist(u,v) = Min\{K(W) | W = (v_0 = u, v_1, \ldots, v_k = v)$ ist einfacher Weg von u nach v $\}$, sofern es einen Weg* u ○——→ ○ v *gibt. $Dist(u,v) = \infty$, wenn es keinen Weg* u ○——→ ○ v *gibt.*
$Dist(v,v) = 0$

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



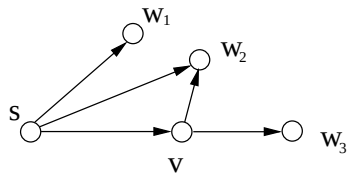
Ist ein kürzester Weg von z nach v, so sind alle

Wege 

oben auch kürzeste Wege. Deshalb ist es sinnvoll, das single source shortest path-Problem zu betrachten, also alle kürzesten Wege von einem Ausgangspunkt zu suchen. Ist s unser Ausgangspunkt, so bietet es sich an, eine Kante minimaler Kosten von s aus zu betrachten:

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

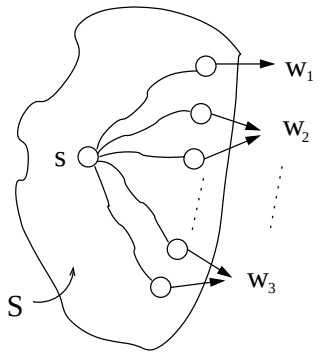
Also, warum ist der Weg s ○——→○ v ein kürzester Weg? (Nachfolgende Aussage gilt bei negativen Kosten so nicht:) Jeder andere Weg von s aus hat durch seine erste Kante s ○——→○ w mindestens die Kosten $K(s, v)$.
Der *Greedy-Ansatz* tut es am Anfang. Wie bekommen wir einen weiteren kürzesten Weg von s aus?



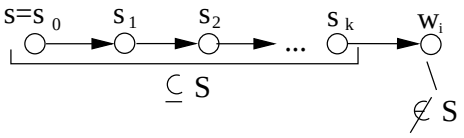
Wir schauen uns die zu s, v adjazenten Knoten an. Oben w_1, w_2, w_3. Wir ermitteln

- 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 kürzesten Weg s ○——→○ 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 alleKnoten 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 ○⟶○ w_i.

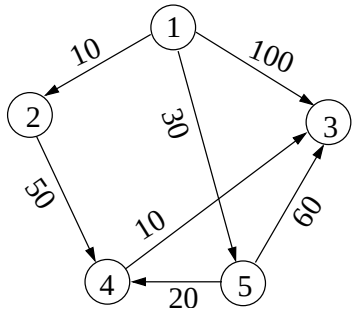
Wie vorher sieht man, dass es wirklich keinen kürzeren Weg s ○⟶○ w_i gibt.
Das Prinzip: Nimm immer den nächstkürzeren Weg von s ausgehend.

8.1 Algorithmus (Dijkstra 1959)

Eingabe: $G = (V, E), K : E \longrightarrow \mathbb{R}^{\geq 0}$(!)
$|V| = n, s \in V$
Ausgabe: array $D[1, \ldots, n]$ of real mit $D[v] = Dist(s, v)$ für $v \in V$.
Datenstrukturen:
S = Menge der Knoten, zu denen kürzester Weg gefunden ist
$Q = V \setminus S$

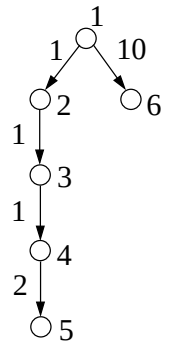
```
1. D[s] = 0, S = {s}, Q=V\{s}
2. for (i = 1 to n - 1){                    //Suchen noch n-1 kürzeste Wege
3.    M = {{v, w}|v ∈ S, w ∈ Q}
4.    for each w ∈ Q adjazent zu S{
5.       D[w] = Min{D[v] + K(v, w)|(v,w) ∈ M}}
6.    w = ein w ∈ Q mit D[w] minimal;
7.    S = S ∪ {w}, Q = Q\{w}}
```

Korrektheit mit 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 $\Pi[u] = v \Longleftrightarrow$ Ein kürzester s ○⟶○ v ist $(s, \ldots, u, v)$.
Π kann leicht in 6. mitgeführt werden. Es ist das Vaterarray des kürzesten-Wege-Baumes mit Wurzel s.



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:



S	Kürzester-Wege-Baum
{1}	
{1, 2}	$\Pi[6] = 1, \Pi[2] = 1, \Pi[3] = 2$
{1, 2, 3}	$\Pi[4] = 5, \Pi[5] = 4$
{1, 2, 3, 4, 5}	
{1, 2, 3, 4, 5, 6}	

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 zu M gehören. Q und S sind als boolesche Arrays mit $Q[v] = true \Longleftrightarrow v \in Q$ 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

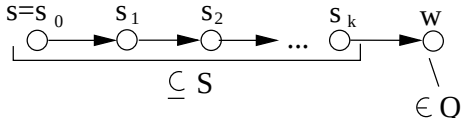
- Delete(u) = Löschen

(Ist die Grundmenge nicht als Indexmenge geeignet: Hashing oder Suchbaum.)
In Q ist ein dictionary implementiert. Jede Operation erfolgt in $O(1)$.
Ziel: bessere Implementierung
Für welche $v \in Q$ kann sich $D[v]$ in 5. nur ändern? Nur für diejenigen, die adjazent zu dem w des vorherigen Laufes sind. Dann sieht es etwa so aus:

8.2 Algorithmus (Dijkstra ohne mehrfache Berechnung desselben D[w])

```
1. D[s] = 0; D[w] = K(s,w) für w ∈ Adj[s];
     D[v] = ∞ für v ∈ V\({s} ∪ Adj[s]);
     Q = V\{s}; S = {s};
2. for i = 1 to n - 1 {
3.    w = ein w ∈ Q mit D[w] minimal;
4.    S = S ∪ {w}; Q = Q\{w};
5.    for each v ∈ Adj[w]{        //D[v] aussparen für v adjazent zu w
6.      if D[w] + K(w, v) < D[v]{
          D[v] = D[w] + K(w, v) }}
```

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 $D_l[v] \leq D_l[w_l]$,
w_l = das Minimum der l-ten Runde. Damit ändert 6. nichts mehr an $D[v]$ für $v \in S_l$.

Laufzeit:

3. insgesamt $n-1$ - mal Minimum finden
4. insgesamt $n-1$ - mal Minimum löschen
5., 6. insgesamt $O(|E|)$, da dort jede Adjazenzliste nur einmal durchlaufen wird.

Mit Q als boolesches Array:

3. $O(n^2)$ insgesamt. Das Finden eines neuen Minimums nach dem Löschen dauert $O(n)$.
4. $O(n)$ insgesamt, 1. Löschen $O(1)$
5., 6. Bleibt bei $O(|E|)$.
Also alles in allem $O(n^2)$.

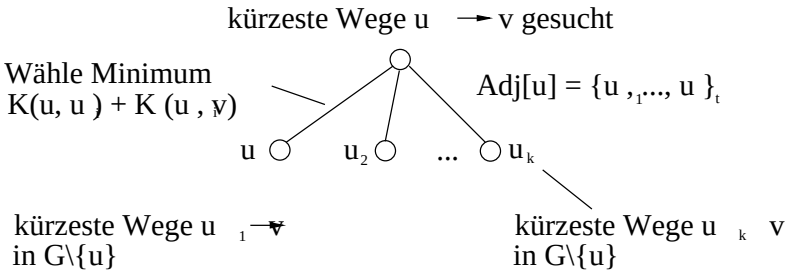
Aber mit Q benötigen wird die klassischen Operationen der priority queue, also Q als heap, den Schlüsselwert aus D.
3. $O(n)$ insgesamt. 4. $O(n \cdot logn)$ insgesamt.
5., 6. $|E|$-mal heap anpassen, mit Decreasekey(v, t) (vgl. Prim) $O(|E| \cdot logn)$. Zum Finden Index mitführen!

Also: Q als boolesches Array:
$O(n^2)$ (Zeit fällt beim Finden der Minima an).

Q als heap mit Index:
$O(|E|logn)$ (Zeit fällt beim Decreasekey(v, t) an).

Ist $|E|logn \geq n^2$, also ist der Graph sehr dicht, dann ist ein Array besser (vgl. Prim).
Ab jetzt betrachten wir wieder eine beliebige Kostenfunktion $K : E \longrightarrow \mathbb{R}$.

Der Greedy-Ansatz wie bei Dijkstra geht nicht mehr. Kürzeste Wege u ○——→ ○ v findet man durch Durchprobieren.
Die Zeit ist $(n-2)! \geq 2^{\Omega(logn+n)} \gg 2^n$!
Es werden im Allgemeinen viele Permutationen generiert, die gar keinen Weg ergeben. Das vermeidet backtracking:



Dies ist korrekt, da Teilwege kürzester Wege wieder kürzeste Wege sind und wir mit kürzesten Wegen immer nur einfache Wege meinen.
Das Ganze ist leicht durch Rekursion umzusetzen:

Eingabe: $G = (V, E), K : E \Longrightarrow \mathbb{R}$ beliebig.

```
KW (W, u, v)                    //u, v ∈ W, W = nach b betrachtete Knotenmenge
1. if (u == v)
     return 0;
2. l = ∞;
3. for each w ∈ Adj[u] ∩ W{
     l' = KW(W\ {u}, w, v);
     l' = K(u, w) + l'_j;
     if(l' < l)
       l = l';                  //Hier Π[w] = u gibt kürzeste Wege selbst
   }
4. return l                     //Ausgabe ∞, wenn Adj[u] ∩ W = ∅.
```

Aufruf: KW(V, a, b)
Korrektheit: Induktion über $|W|$.

Etwas einfacher ist es, alle einfachen Wege u○——→○v systematisch zu erzeugen und den Längenvergleich nur am Ende durchzuführen.
Datenstruktur: L
L = Liste von Knoten als Keller implementiert
Intention: L enthält den Weg vom Startknoten a bis zum aktuellen Knoten.

G = Liste von Knoten. Der aktuell kürzeste Weg a○——→○b
L, G = globale Arrays

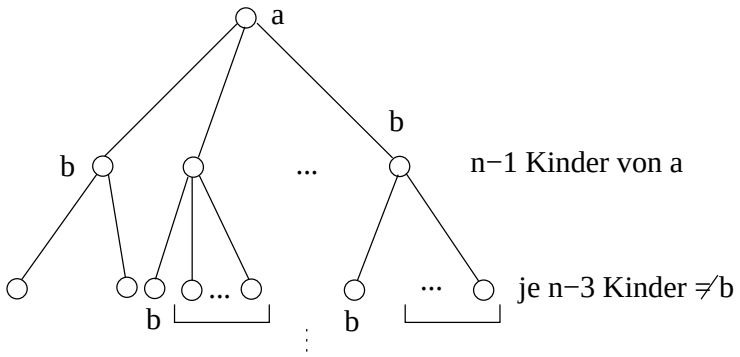
Aufruf mit K(L) = ∞, K(G) = ∞, L = (a), KW(v, a, b)

```
KW(W, u, v){
1. if (u == v){
     if (K(L) < K(G)) // K(L), K(G) = Kosten von L, G
       G = L;
     return;
   }
2. for each w ∈ Adj[u] ∩ W {
     w auf Keller L ablegen;
     KW(W\{u}, w, v);
     w vom Keller L löschen
   }
}
```

Korrektheit mit der Aussage: Beim Aufruf von KW(W, w, v) ist L = w…a.
Am Ende des Aufrufs KW(W, w, v) enthält G einen kürzesten Weg der Art $G = b \ldots L$. (L ist das L von oben)
Das Ganze induktiv über $|W|$.

Laufzeit der rekursiven Verfahren?

Enthält der Graph alle $n(n-1)$ Knoten, so sieht der Aufrufbaum folgendermaßen aus:



Also $\geq (n-2)(n-3) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 \geq 2^{\Omega(n \cdot logn)}$.

Wieviele verschiedene Aufrufe haben wir höchstens? Also Aufrufe KW(W, c, d)?

- $W \subseteq V \leq 2^n$ Möglichkeiten.
- $c, d \leq n$ Möglichkeiten, da Endpunkt immer gleich.

Also $n \cdot 2^n = 2^{logn} \cdot 2^n = 2^{O(n)}$.
Also bei $2^{\Omega(nlogn)}$ Aufrufen wie oben viele doppelt. Es ist ja $\frac{2^{O(n)}}{2^{\Omega(nlogn)}} = \frac{1}{2^{\Omega(logn)}} = \frac{1}{n^\epsilon} \longrightarrow 0$ (wobei $n^\epsilon = n^{\Omega(1)}$) für ein $\epsilon > 0$ konstant.
Also: Vermeiden der doppelten Berechnung durch Merken (Tabellieren).
Dazu das Array $T\overbrace{[1 \ldots 2^n]}^{\in\{0,1\}^n}[1 \ldots n]$ of int mit der Interpretation: Für $W \subseteq V$ mit $b, v \in W$ ist $T[W, v]$ = Länge eines kürzesten Weges v ○⟶○ b nur durch W.

Füllen $T[W, v]$ für $|W| = 1, 2, \ldots$

1. $|W| = 1$, dann $v = b \in W, T[\{b\}, b] = 0$
2. $|W| = 2$, dann $T[W, b] = 0, T[W, v] = K(v, b)$, wenn $v \neq b$
3. $|W| = 3, T[W, b] = 0$

T[W, v] ergibt sich aus:

$l = \infty, W = W \setminus \{v\}$

for each $u \in$ Adj[v]$\cap$W{
  $l' = K(v,u) + T[W', u]$
  if $(l' < l)$
    $l = l'$
}
$T[W, v] = l$

Also, das heißt:

$T[W, v] = Min\{Q(v,u) + T[W', u] | W' = W \setminus \{v\}, u \in W\}$

$T[W, v] = \infty$, wenn keine Kante v ○——→○ u mit $u \in W$.
Das Mitführen des Weges durch $\Pi[W, v] = u, u$ vom Minimum ermöglicht eine leichte Ermittlung der Wege. Also, der Eintrag $T[W, v]$ wird so gesetzt:

```
Setze(W, v, b) {
1. if (v == b) {
     T[W, v] = 0;
     Π[W, v] = b;
     return; }
   W' = W \ {v}; // Hier |W| ≥ 2
   T[W,v] = ∞;
2. for each u ∈ Adj[v] ∩ W' {
     l = K(v,u) + T[W', u];
     if (l < T[W,v]) {
       T[W,v] = l;
       Π[W,v] = u;
     }}}
```

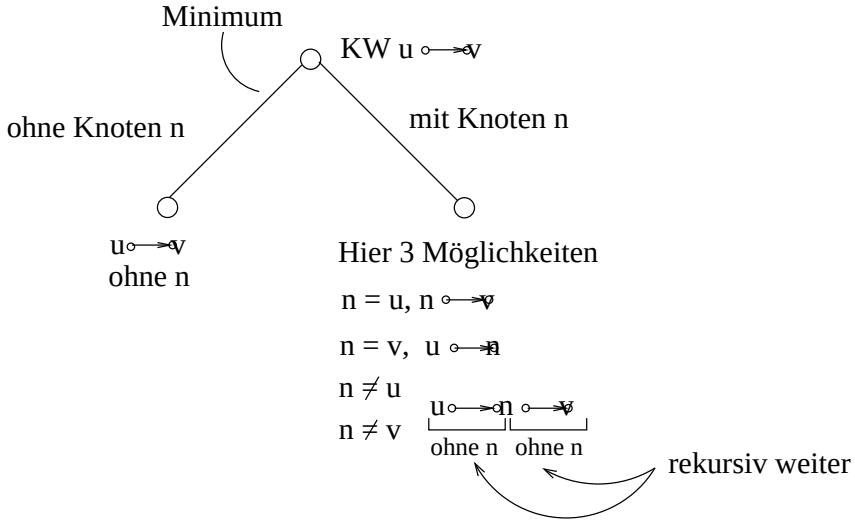
Dann insgesamt

```
KW(V, •, b) {
1. for i = 1 to n {
2.    for each W ⊆ V, b ∈ W, |W| = i{
3.       for each v ∈ W{
4.          Setze(W, v, b);
          }}}}
```

Dann enthält $T[V, a]$ das Ergebnis. Weg ist $a_0 = a, a_1 = \Pi[V, a], a_2 = \Pi[V \setminus \{a\}, a_1], \ldots, a_i = \Pi[V \setminus \{a_0, \ldots, a_{i-2}\}, a_{i-1}], \ldots$
Invariante für 1.
Nach dem $l-$ten Lauf ist $T_l[W, v]$ korrekt für alle W mit $|W| \leq l$.
Laufzeit: $O(n^2 \cdot 2^n)$, da ein Lauf von *Setze(W, v)* in $O(n)$ möglich ist.

Das hier verwendete Prinzip:
Mehr rekursive Aufrufe als Möglichkeiten $\Rightarrow$ Tabellieren der Aufrufe heißt:

Dynamisches Programmieren (Auffüllen einer Tabelle, 1950er)
Wir beginnen einmal mit einer anderen Fallunterscheidung beim backtracking:

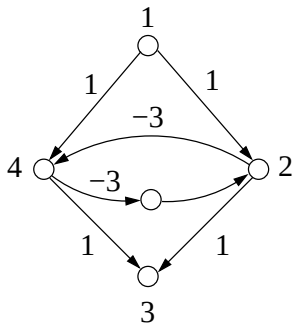


Korrektheit: Ist $u, v + n$ und ist ein kürzester Weg u ○⟶○ v ohne n, dann wird dieser nach Induktionsvoraussetzung links gefunden. Erhalte jetzt jeder kürzeste Weg u ○⟶○ v den Knoten n. Wird ein solcher unbedingt rechts gefunden?

Nur dann, wenn die kürzesten Wege u ○⟶○ n, n ○⟶○ v keinen weiteren gemeinsamen Knoten haben. Wenn sie einen gemeinsamen Knoten w haben,

so u ○⟶ w ○⟶ n ○⟶ w ○⟶ v. Da dieser Weg kürzer ist als

jeder Weg ohne n, muss w ○⟶ n ○⟶ w ein Kreis der Länge < 0 sein. Betrachten wir also jetzt $K : E \longrightarrow \mathbb{R}$, aber so, dass keine Kreise < 0 existieren.



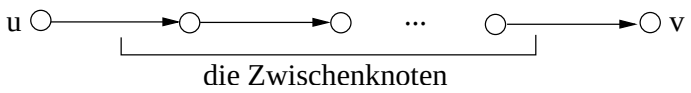
kürzester Weg 1 ○→○ 3 ohne 4, Kosten 2

kürzester Weg 1 ○→○→○ 4 Kosten −2 (2)

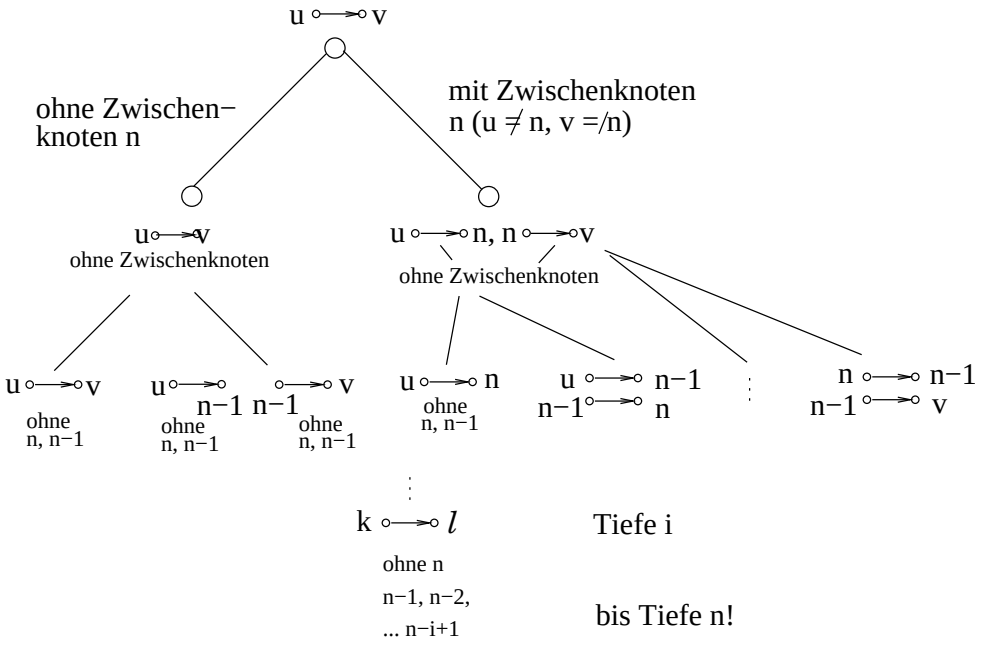
kürzester Weg 4 ○→○→○ 3 Kosten −2 (2)

und (2, 4, ○2) Kreis der Kosten −6

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



Also Backtracking:



Also rekursive Prozedur:

KW(u, v, i) für kürzeste Wege u○——→○v ohne Zwischenknoten $n, n-1, n-$

$1, \ldots, n-i+1$. Kürzester Weg ergibt sich durch KW(u, v, 0). Rekursionsanfang KW(u, v, n) gibt Kante u○⟶○v. Wieviele verschiedene Aufrufe? $O(n^3)$!

Also dynamisches Programmieren:

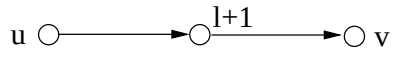
$T[u, v, i]$ = kürzester Weg u○⟶○v wobei Zwischenknoten $\subseteq \{1, \ldots, i\}$ (Also nicht unter i+1, ..., n.)
0. $T[u, v, 0] = K(u, v)$ für alle u, v!
1. $T[u, v, 1] = Min\{T[u, 1] + T[1, v], T[u, v, 0]\}$ für alle u, v
⋮
n. $T[u, v, n] = Min\{T[v, n, n-1] + T[n, v, n-1], T[u, v, n-1]$ für alle u, v
All pairs shortest path.

8.3 Algorithmus Floyd Warshall

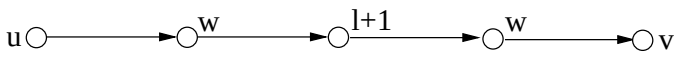
G ohne Kreise ¡ 0, V = {1, ..., n}

```
1. T[u, v] = 0, T[u, v] = K(u, v) für (u, v) ∈ E
   T[u, v] = ∞ für u ≠ v, (u, v) ∉ E
2. for i = 1 to n{
     for each (u, v) ∈ V{        Alle geordneten Paare.
       T[u, v] = Min{T[u, v], T[u, i] + T[i, v]}
   }
```

Invariante: Nach $l-$tem Lauf der Schleife in 1. entsteht $T_l[u, v]$ = Länge eines kürzesten Weges u○⟶○v mit Zwischenknoten $\subseteq \{1, \ldots, l\}$.
Wichtig: Keine negativen Kreise, denn in l+1-ter Runde gilt $T_l[u, l+1]+T_l[l+1, v] < T_l[u, v]$,
dann ist der Weg hinter $T_l[u, l+1] + T_l[l+1, v]$ ein einfacher!

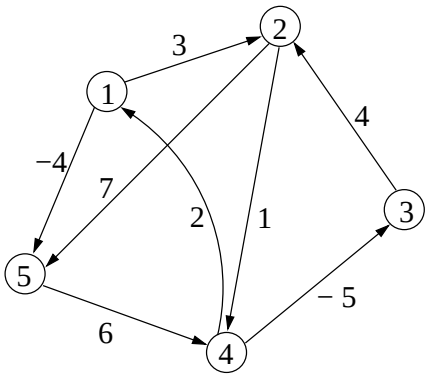


kürzer als kürzeste Wege u○⟶○v ohne $l+1$, dann oben nicht



da sonst K(u ○⟶ ○ ⟶ ○ v) <0 sein müsste.
Erkennennegative Kreise: Immer gilt, also bei beliebiger Kostenfunktion, dass Floyd

Warshall die Kosten eines Weges von u ○⟶○ v in $T[u, v]$ leitet. Dann $T[u, v] < 0 \iff G$ hat Kreis < 0.
Laufzeit: $O(n^3)$ (Vergleiche Dijkstra $O(|E|log|V|)$ oder $O(|V|^2)$.)



Am Anfang

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

Bevor k auf 2 geht:

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

Es ist A[i, j] = Min{A[i, j], A[i, 1] + A[1, j]}
Direkte Wege.
Wege mit Zwischenkomponenten 1.
In 1 bevor k auf 3 geht:
Direkte Wege + Wege mit Zwischenknoten 1 + Wege mit Zwischenknoten 1, 2.
Nicht: mit 2 Zwischenknoten!

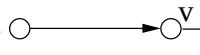
9 Flüsse in Netzwerken

$G = (V, E)$ gerichtet, Kostenfunktion $K : E \longrightarrow \mathbb{R}^{\geq 0}$. Hier nun $K(u, v) =$ die Kapazität von (u, v). Stellen uns vor, (u, v) stellt eine Verbindung dar, durch die etwas fließt (Wasser, Strom, Autos, ...).
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* $K : V \times V \longrightarrow \mathbb{R}^{\geq 0}, K(u, v) = 0$ *falls* $(u, v) \notin E$.
Außerdem gibt es zwei ausgezeichnete Knoten

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

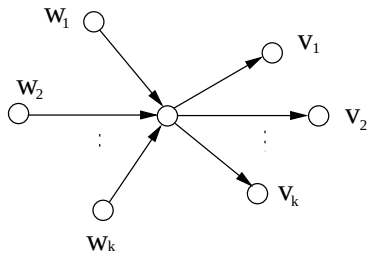
Wir verlangen noch: Jeder Knoten $v \in V$ *ist auf einem Weg* 

.

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

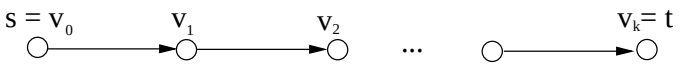
$\in E$ ist $f(u, v) \in \mathbb{R}^+$. Es muss für einen Fluss f gelten:

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

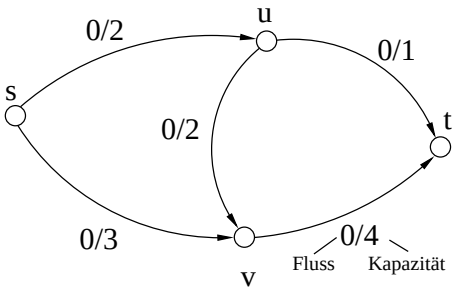


dann $\sum f(w_i, u) = \sum(w, v_i)$.

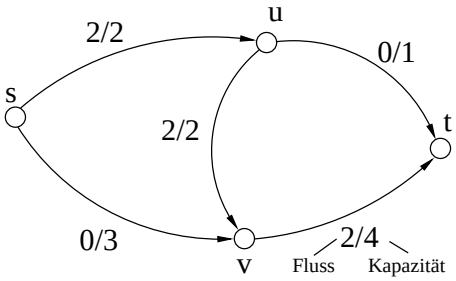
Prinzip: Methode der Erweiterungspfade (Ford - Fulkerson 1950er Jahre)
1. Beginne mit dem Fluss 0, f(u, v) = 0 für alle (u, v)

2. Suche einen Weg (Erweiterungspfad) 

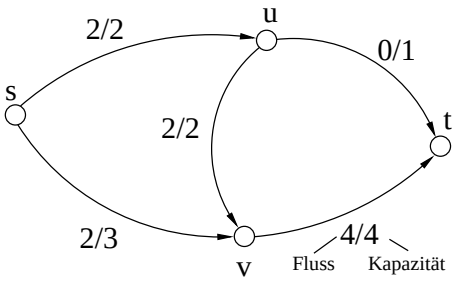
so dass für alle $f(v_i, v_{i+1}) < K(v_i, v_{i+1})$.
3. Erhöhe den Fluss entlang des Weges so weit es geht. Dann bei 2. weiter.
Wieso negative Flüsse?



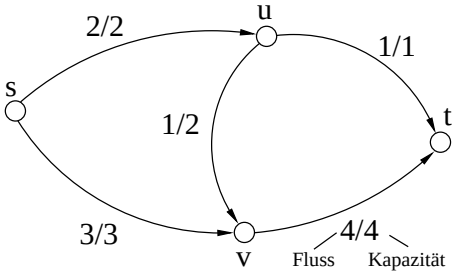
Weg (s, u, v, t) + 2



Weg (s, v, t) + 1

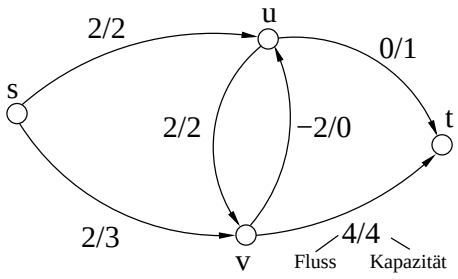


kein Erweiterungspfad, aber



ist größer!

Mit negativen Flüssen:

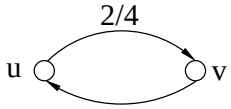


Immer ist $f(u,v) = -f(v,u)$.
Weg (s, v, u, t) + 1 gibt den minimalen Fluss.
Wir lassen auch $f(u,v) < 0$ zu.

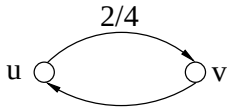
Definition 9.2: *Ein Fluss ist eine Funktion* $f : V \times V \longrightarrow \mathbb{R}$ *mit*

- $f(u,v) \leq K(u,v)$ *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: Maximaler Fluss $|f|$.

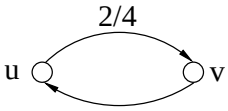
- Immer ist $f(u,u) = -f(u,u) = 0$, da $K(u,u) = 0$, da nie $(u,u) \in E$.
- Auch $f(u,v) = f(v,u) = 0$, wenn $(u,v), (v,u) \notin E$. Denn es ist $f(u,v) = -f(v,u)$, also ein Wert > 0.
- Ist $f(u,v) \neq 0$, so u○⟶○v $\in E$ oder v○⟶○u $\in E$.



- Nicht sein kann wegen $f(u,v) = -f(v,u)$. Hier würden wir setzen $f(u,v) = f(v,0) = 0$.



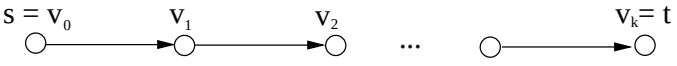
oder



Wir können (müssen) kürzen, aber geht.

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

- *Das Restnetzwerk G_f mit Restkapazität K_f ist gegeben durch:*
  $K_f = (u, v) = K(u, v) - f(u, v) \geq 0, E_f = \{(u, v) | K_f(u, v) > 0\}$. *(Es ist $K_f(u, v) \geq 0$, da $f(u, v) \leq K(u, v)$)*
  $G_F = (V, E_f)$
- *Ein Erweiterungspfad von G und f ist ein einfacher Weg in G_f*

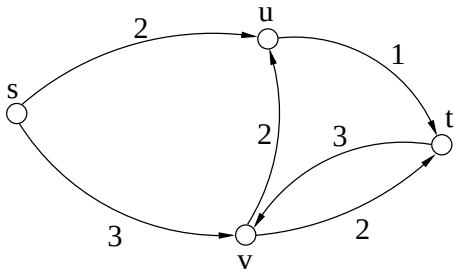
  $W =$ 

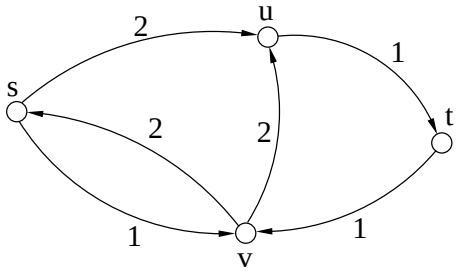
  Die Restkapazität von W ist $K_f(W) = Min\{K_f(v_i, v_{i+1})\}$ (Nach Definition gilt $K_f(v_i, v_{i+1}) > 0$))

Es ist G_f ein Flussnetzwerk und es ist $g : V \times V \longrightarrow \mathbb{R}$ mit $g(v_i, v_{i+1}) = K_f(W) > 0$, $g(v_{i+1}, v_i) = -K_f(W) = -G(v_i, v_{i+1})$, $g(u, v) = 0$ für $(u, v) \notin W$, ein zulässiger Fluss auf $G_f \cdot |g| = K_f(W)$.
Tatsächlich gilt nun sogar:

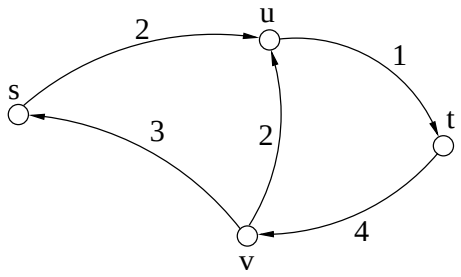
Lemma 9.1: *Sei G, K, f Flussnetzwerk mit zulässigem Fluss f. Sei G_f das Restnetzwerk. Sei g irgendein zulässiger Fluss auf G_f. Dann ist $f+g$ Fluss auf G, $|f+g| = |f| + |g|$.*

Restnetzwerke zu vorangegangenem Netzwerk





Kapazitäten immer ≥ 0



kein Erweiterungspfad.

Beweis. Wir müssen also überlegen, dass $f + g$ zulässiger Fluss von G ist. Kapazitätsbedingung: s ist $g(u,v) \leq K_f(u,v) = K(u,v) - f(u,v)$.
Also $(f+g)(u,v) = f(u,v) + g(u,v) \leq f(u,v) + K(u,v) - f(u,v) = K(u,v)$
Symmetrie
$(f+g)(u,v) = f(u,v) + g(u,v) = -f(v,u) - g(v,u) = -(f+g)(v,u)$ Kirchhoff: Sei $u \in V \setminus \{s,t\}$, dann $\sum\limits_{v \in V} (f+g)(u,v) = \sum\limits_{v \in V} (f(u,v) + g(u,v))$
$= \sum\limits_{v \in V} f(u,v) + \sum\limits_{u \in V} g(u,v) = 0.$
Schließlich ist $|f+g| = \sum\limits_{v \in V} (f+g)(s,v) = |f| + |g|$. □

9.1 Algorithmus (Ford Fulkerson)

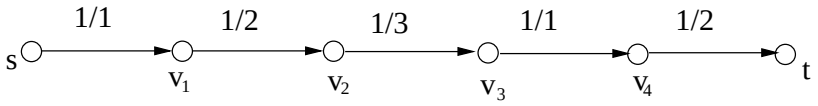
1. $f(u,v) = 0$ für alle $u, v \in V$
2. while Es gibt Weg s ○⟶○ t in G_f {
3. W = ein Erweiterungspfad in G_f
4. g = Fluss in G_f mit $g(u,v) = K_f(W)$ für alle u ○⟶○ v $\in W$, wie oben
5. $f = f + g$ }

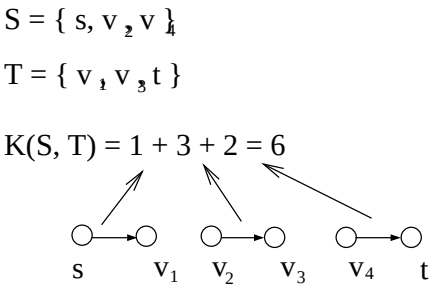
Gib f als maximalen Fluss aus.

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

- *Kapazität von S,T, $K(S,T) = \sum\limits_{u \in S, v \in T} K(u,v)$ mit $K(u,v) \geq 0, u \in S, v \in T$*
- *Ist f ein Fluss, so ist $f(S,T) = \sum\limits_{u \in S, v \in T} f(u,v)$.*

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





$F(S,T) = 1 - 1 + 1 - 1 + 1 = 1$ (Betrag des Flusses)
S, T in mehreren Stücken $\longrightarrow$ kein minimaler Schnitt.

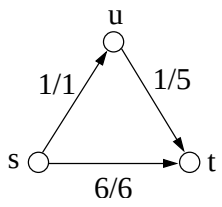
Tatsächlich gilt sogar für jeden Schnitt S, T, dass $f(S,T) = |f|$
Induktion über $|S|$. $|S| = 1$, dann $S = \{s\}$, dann gilt es. Sei $|S| = l+1, v \in S \setminus \{s\}$, sei $S' = S \setminus \{v\}, T' = \cup\{v\}$. Dann $f(S,T) = f(S',T') + \sum_{u \in S} f(u,v) + \sum_{u \in T} f(v,u) = |f| + \sum_{u \in V} f(v,u) = |f|$.
Also: Für jeden Schnitt $|f| \leq K(S,T)$.
Also auch $Max\{|f||fFluss\} \leq Min\{K(S,T)|S,TSchnitt\}$ Es gibt einen Fluss f^*, der diese obere Schranke erreicht, $f^* = Min\{K(S,T)|S,TSchnitt\}$:

Satz 9.1(Min-Cut-Max-Flow): *Ist f ein zulässiger Fluss in G. Äquivalent sind (1), (2) und (3).*
(1) f ist maximaler Fluss.
(2) G_f hat keinen Erweiterungspfad.
(3) Es gibt einen Schnitt S,T, so dass $|f| = K(S,T)$.
(Immer $f(S,T) = |f|, K(S,T) \geq |f|$)

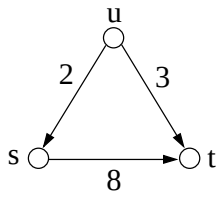
Beweis. (1) $\longrightarrow$ (2) gilt, da f maximal. Gälte (2) nicht, dann gälte auch (1) nicht.
(2) $\longrightarrow$ (3) Setze $S = \{u \in V|$ Es gibt Weg (s, u) in $G_f\}$
$T = V \setminus S$
Dann $s \in S$ und $t \in T$, da kein Erweiterungspfad nach (2). Also S,T ist ordentlicher Schnitt. Für $u \in S, v \in T$ gilt:
$O = K_f(u,v) = K(u,v) - f(u,v)$
Also $K(u,v) = f(u,v)$.
Dann $|f| = f(S,T) = \sum_(u \in S, v \in T) f(u,v) = \sum_(u \in S, v \in T) K(u,v) = K(S,T)$.
(3) $\longrightarrow$ (1) gilt, da immer $|f| \leq K(A,B)$. □

Korrektheit von Ford-Fulkerson:
Invariante:
f_l ist zulässiger Fluss Quintessenz: Am Ende ist f_l maximaler Fluss (Mincut-max-flow).
Termination: $f_l(S,T)$ erhöht sich jedesmal.
Laufzeit von Ford Fulkerson?
Bei ganzzahligen Kapazitäten reicht $O(|E| \cdot |f^*|)$, wobei f^* ein maximaler Fluss ist.
Rationale Zahlen: Normieren!

Min. Schnitt = Max. Fluss



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



$S = \{s\}, T = \{u, t\}, K(S,T) = 8$
$S = \{s, u\}, T = \{t\}, K(S,T) = 11$
$|f| \leq 8$. Vergleiche hierzu Seite 122