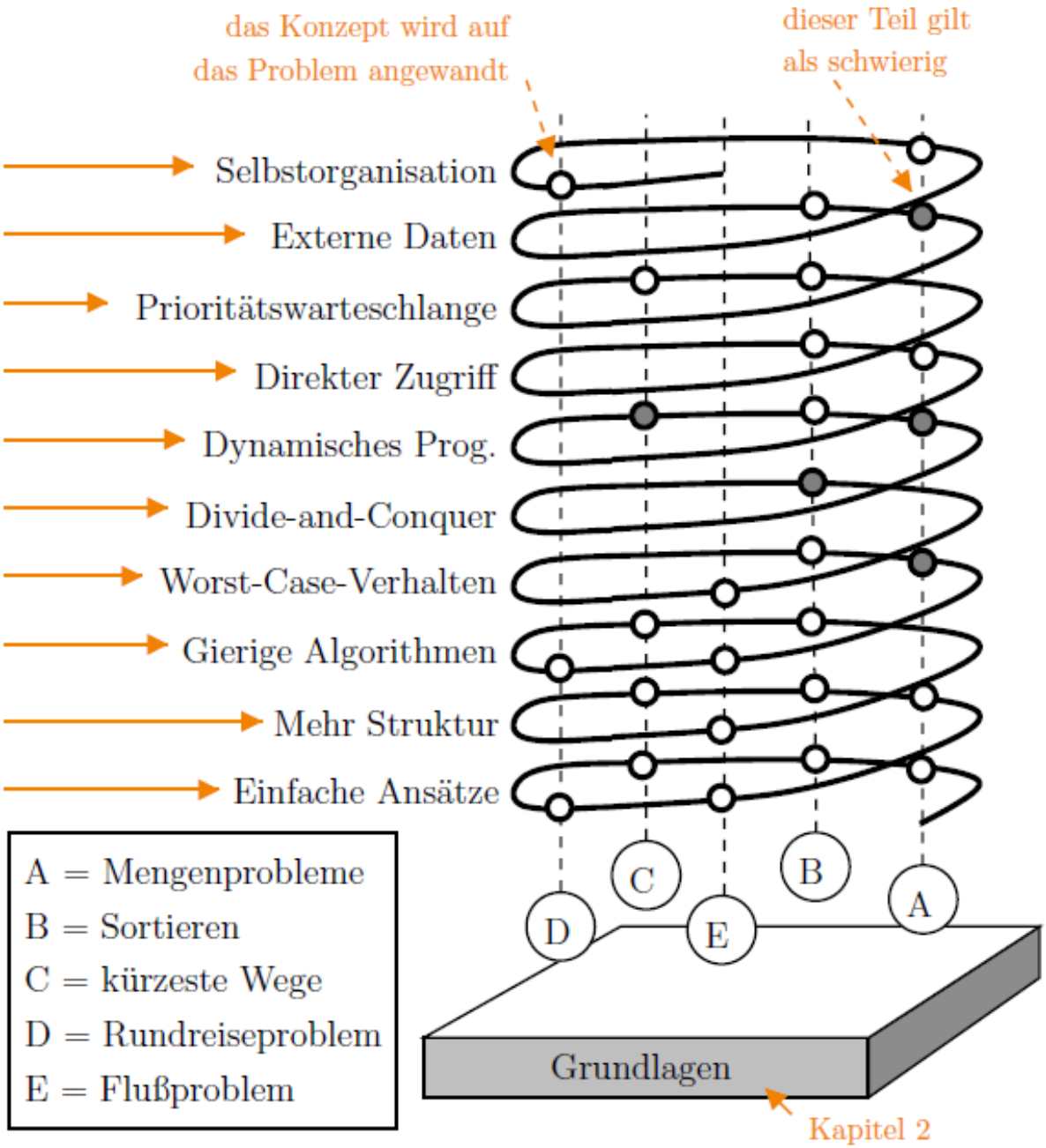
Einordnung des Problems (Maximaler Fluss)



1.6 Das Flussproblem

Die Tatsache, dass in den Läden der Firma von Algatha Online-Rollenspiele mit virtueller Realität gespielt werden, erfordert eine enorme Bandbreite des firmeneigenen Netzwerks, mit dem alle Filialen untereinander verbunden werden. Daher interessiert die Firmeneigner natürlich, ob mit wachsender Zahl der Filialen noch alle Spielplätze in den Filialen bedient werden können, oder ob das firmeninterne Netzwerk weiter auszubauen ist.

Daher wird das Firmennetzwerk ebenfalls als Graph modelliert. Die Knoten werden durch die Filialen, den zentralen Server und die Router und Switches im Netzwerk bestimmt. Die Kanten entsprechen den Leitungen. Um den Fluss der Daten durch das Netzwerk berechnen zu können, werden für jede Leitung zwei Angaben benötigt: die Kapazität als Datenmenge, die pro Zeiteinheit in die Leitung eingespeist werden kann, und die Dauer, die ein Datenpaket benötigt, um am Ende der Kante anzukommen.

Auweia! Das klingt schon umgangssprachlich kompliziert und wird in der Definition vermutlich nicht leichter... Klar, wenn viele Spieler gleichzeitig unsere Online-Spiele verwenden, kann es zu Engpässen kommen – und die sollten wir durch gute Planung vermeiden. Deshalb berechnen wir vorher, wie viel maximal über unser internes Datennetz fließen kann. Angenommen am zentralen Server stehen permanent Datenpakete zum Verschicken bereit. Dann ist die Frage, wie viele Datenpakete zum Beispiel innerhalb von einer Stunde an den Filialen ankommen können.

Definition 1.21: Dynamischer Maximaler Fluss

Sei ein schleifenfreier Graph $G = (V, E)$ mit der Kapazität pro Kante $kap : E \to \mathbb{N}_0$ und der Dauer pro Kante $\tau : E \to \mathbb{N}_0$ gegeben. Weiterhin seien $S \subset V$ die Startknoten ($S \neq \emptyset$) und $T \subset V$ die Zielknoten ($T \neq \emptyset$) zwei disjunkte Mengen ($S \cap T = \emptyset$). Der Zeithorizont $tmax \in \mathbb{N}$ bestimmt die Anzahl der betrachteten Zeitschritte.

Dann berechnet die Operation DYN-MAX-FLUSS für jeden Zeitschritt die Datenmenge

$$f : E \times \{0, \ldots, tmax\} \to \mathbb{N}_0,$$

die maximal auf jeder Kante transportiert werden kann. Hierfür müssen die folgenden Bedingungen gelten:

- Die Kapazität der Kanten wird nicht überschritten.

$$\forall e \in E \; \forall t \in \{0, \ldots, tmax\} : \; f(e,t) \leq kap(e)$$

- Von jedem inneren Knoten, also nicht Start- oder Zielknoten, ist höchstens so viel abgeflossen, wie angekommen ist.

$$\forall x \in V \setminus (S \cup T) \; \forall t \in \{0, \ldots, tmax - 1\} : \; aktuell(x,t) \geq 0,$$

wobei *aktuell* die in einem Zeitschritt an einem Knoten verbliebenen Datenmenge bezeichnet:

$$aktuell(x,t) = \sum_{e=(v,x) \in E} \sum_{i=0}^{t-\tau(e)} f(e,i) - \sum_{e=(x,w) \in E} \sum_{i=0}^{t} f(e,i).$$

Die linke Doppelsumme misst die Datenmenge, die bis zum Zeitpunkt t am Knoten x angekommen ist, und die rechte Doppelsumme misst die bis zum Zeitpunkt t vom Knoten x abgeflossene Datenmenge. Bei den ankommenden Daten werden, abhängig von der Durchlaufzeit $\tau(e)$ jeder Kante e Daten nicht berücksichtigt, die noch unterwegs sind.

- Alle losgeschickten Daten sind am Ende bei den Zielknoten.

$$\sum_{x \in T} aktuell(x, tmax) = - \sum_{x \in S} aktuell(x, tmax)$$

- Der Gesamtfluss ist maximal.

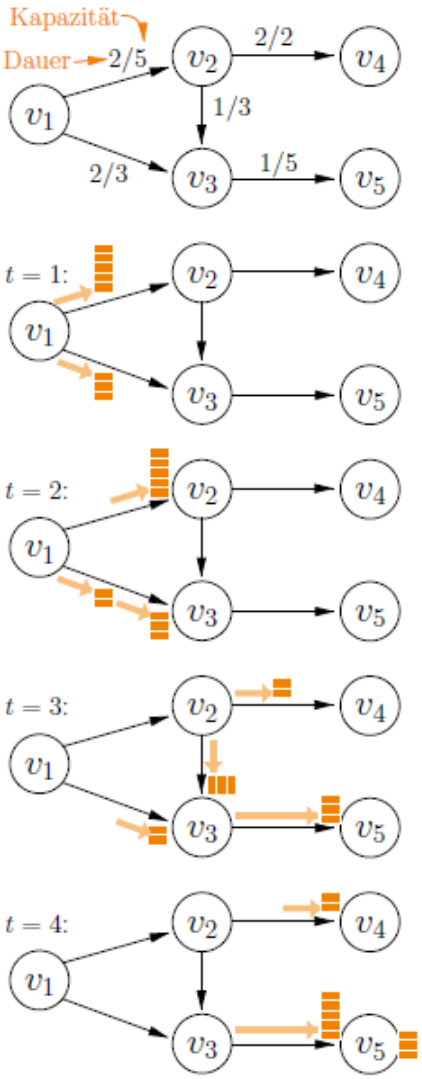
$$\sum_{x \in T} aktuell(x, tmax) \text{ ist maximal}$$

Das habe ich jetzt verstanden – das Umgangssprachliche ist ja nur immer in eine Formel gegossen. Und die Gleichung mit den beiden Doppelsummen ist natürlich auch notwendig, sonst müsste man sich ja fragen »Wenn drei Leute aus einem leeren Zimmer kommen, wie viele müssen dann wieder hineingehen, dass keiner drin ist?«

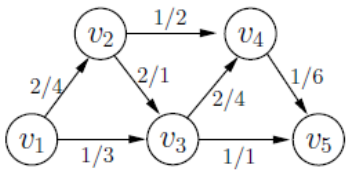
Beispiel 1.22:

Das auf der nächsten Seite am Rand abgebildete Flussproblem mit dem Startknoten $S = \{v_1\}$, den Zielknoten $T = \{v_4, v_5\}$ und dem Zeithorizont $tmax = 4$ besitzt den maximalen Fluss von 10 Einheiten. Ein möglicher Ablauf ist in der Abbildung dargestellt. Interessant ist dabei die Verbindung zwischen v_1 und v_3 – dort sind zum Zeitpunkt $t = 2$ zwei »Ladungen« hintereinander unterwegs. Obwohl die Gesamtmenge auf der Verbindung die zulässige Kapazität überschreitet, ist dies korrekt, da sich die Kapazität in der Definition immer auf einen (beliebigen) Schnitt durch die Verbindung bezieht: Es darf also in jedem Zeitschritt die maximal erlaubte Menge losgeschickt werden. □

Die vermutlich erste »offizielle« Erwähnung eines Maximaler-Fluss-Problems war von Ford & Fulkerson (1956). Eine lesenswerte Erläuterung der Hintergründe findet man bei Schrijver (2002). Eine unserer Definition ähnliche Problemstellung wurde auch bereits von Ford & Fulkerson (1962) formuliert.



Veranschaulichung eines dynamischen maximalen Flusses.



Aufgabe 1.8: Dynamischer maximaler Fluss

Betrachten Sie das nebenstehende Problem für den dynamischen maximalen Fluss. Der Zeithorizont sei $tmax = 6$. Versuchen Sie, möglichst viele Einheiten von v_1 zu v_5 zu bringen.

4.6 Rückführung auf ein einfacheres Problem

Bei der in Abschnitt 4.5.2 vorgestellten Breitensuche hatten wir bereits mit der Vereinfachung des Kürzeste-Wege-Problems argumentiert. In diesem Abschnitt möchten wir einen Schritt weiter gehen. Wir vereinfachen das Problem des dynamischen maximalen Flusses (Def. 1.21) zu einem statischen maximalen Fluss. Wir lösen das originale Problem, indem wir es auf das einfachere Problem zurückführen.

Das bisher behandelte Flussproblem ist insbesondere wegen des Zeitfaktors relativ kompliziert und lässt sich daher auch nur schwer algorithmisch bearbeiten. Daher führen wir eine einfachere Variante ein und betrachten im Weiteren, was die beiden Probleme miteinander zu tun haben.

Definition 4.57: Maximaler Fluss

Sei ein schleifenfreier Graph $G = (V, E)$ mit der Kapazität pro Kante $kap : E \to \mathbb{N}_0$ gegeben. Weiterhin seien $s \in V$ der Startknoten und $t \in V$ der Zielknoten mit $s \neq t$.

Dann berechnet die Operation MAX-FLUSS die Datenmenge

$$f : E \to \mathbb{N}_0,$$

die maximal auf jeder Kante transportiert werden kann. Hierfür müssen die folgenden Bedingungen gelten:

- Die Kapazität der Kanten wird nicht überschritten.

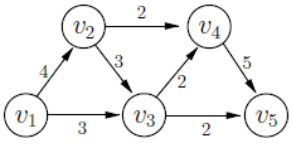
$$\forall e \in E : \; f(e) \leq kap(e)$$

- An jedem inneren Knoten fließt gleich viel ab, wie ankommt.

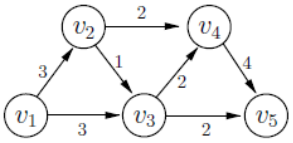
$$\forall x \in V \setminus \{s, t\} : \sum_{e=(v,x) \in E} f(e) = \sum_{e=(x,v) \in E} f(e)$$

- Der Gesamtfluss ist maximal.

$$\sum_{e=(x,t) \in E} f(e) \text{ ist maximal}$$



Ausgangsgraph.

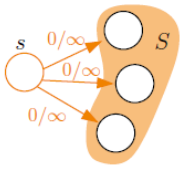


Lösung mit maximalem Fluss.

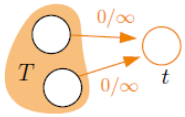
Beispiel 4.58:
Nebenstehend ist ein Graph mit Startknoten v_1 und Zielknoten v_5 für das maximale Flussproblem dargestellt – die Zahlen an den Kanten entsprechen der Kapazität. Darunter ist eine mögliche Lösung mit einem maximalen Fluss von 6 Einheiten dargestellt – die Zahlen an den Kanten entsprechen dem Fluss über die Kante. □

Das Bild von Fahrzeugen, die sich im Straßennetz bewegen, funktioniert hier nicht mehr. Aber ich kann mir so einen Graphen als mit Wasser gefülltes Röhrensystem vorstellen.

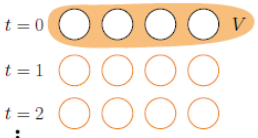
Soll nun eine Instanz des dynamischen maximalen Flussproblems $(G = (V, E), kap, \tau, S, T, tmas)$ mit einem Algorithmus für das maximale Flussproblem gelöst werden, muss zunächst die Probleminstanz in eine Instanz des einfacheren Problems $(G' = (V', E'), kap', s', t')$ überführt werden. Dies geschieht mit den folgenden fünf Schritten.



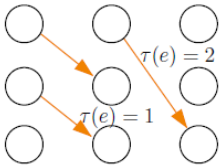
Schritt 1.



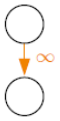
Schritt 2.



Schritt 3.



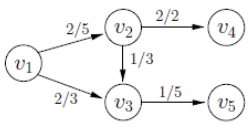
Schritt 4.



Schritt 5.

1. Bei mehreren Quellen ($\#S > 1$) wird V um einen neuen Startknoten s erweitert. Es wird eine Kante $e = (s, x)$ für jeden Knoten $x \in S$ eingeführt und es gilt $kap(e) = \infty$ und $\tau(e) = 0$.
2. Analog wird für mehrere Zielorte ($\#T > 1$) die Menge V um einen neuen Zielknoten t erweitert. Die Kanten $e = (x, t)$ $(x \in T)$ besitzen $kap(e) = \infty$ und $\tau(e) = 0$.
3. Für den so gebildeten Graphen werden jetzt die Knoten vervielfacht – einmal für jeden Zeittakt $0, \ldots, tmax$: Damit erhalten wir die Menge
$$V' = \left\{v^{(t)} | v \in V \wedge t \in \{0, \ldots, tmax\}\right\}$$
4. Für jede Kante $e = (u, v) \in E$ mit Fahrtzeit $\tau(e)$ wird eine Menge an Kanten konstruiert:
$$neuE_{(u,v)} = \left\{(u^{(t)}, v^{(t+\tau(e))} \mid 0 \leq t \leq tmax - \tau(e)\right\}$$
Dann ergibt sich $E'' = \bigcup_{e \in E} neuE_e$, wobei für jede Kante $(u^{(t)}, v^{(t')})$ gilt: $kap'(u^{(t)}, v^{(t')}) = kap(u, v)$.
5. Zum Abschluss müssen wir noch das Liegenlassen von Waren für einen späteren Takt erlauben, indem wir E' wie folgt erweitern:
$$E' = E'' \cup \left\{(v^{(t)}, v^{(t+1)}) | v \in V \wedge t \in \{0, \ldots, tmax - 1\}\right\}$$
Die Kapazität dieser Kanten sei $kap'(v^{(t)}, v^{(t+1)}) = \infty$.

Damit wurde eine Probleminstanz des maximalen Flusses mit $s' = s^{(0)}$ und $t' = t^{(tmax)}$ konstruiert.



Instanz des dynamischen maximalen Flussproblems

Beispiel 4.59:
Das nebenstehende Beispiel eines dynamischen maximalen Flussproblems ist identisch zu dem in Beispiel 1.22 betrachteten Problem. Wenden wir die Konstruktion darauf an, erhalten wir den Graphen in Bild 4.13. Wenn man in diesem Graphen nun den maximalen Fluss sucht, gelangt man beispielsweise bei der Lösung in Bild 4.14 an. Dies ist dieselbe Lösung wie in Beispiel 1.22. □

Wie man am Beispiel leicht erkennt, kann man sowohl die dynamische Lösung leicht im konstruierten Graphen darstellen als auch anders herum. Tatsächlich bietet eine Darstellung wie in Bild 4.14 eine größere Übersichtlichkeit als die Visualisierung in Kapitel 1. Auf einen formalen Beweis für die Transformation der jeweiligen Lösungen ineinander verzichten wir hier.

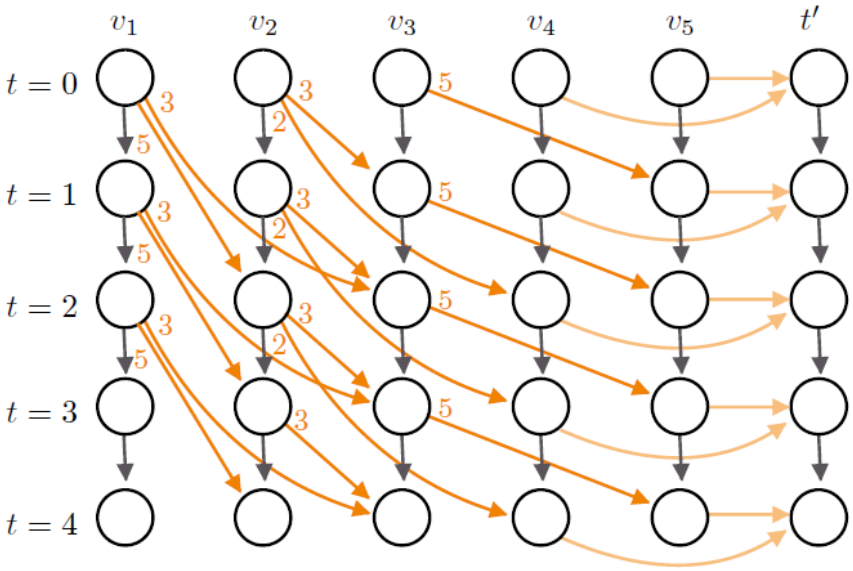


Bild 4.13: Dieser Graph entsteht gemäß der angegebenen Konstruktion aus dem Graphen in Beispiel 4.59. Zur besseren Übersichtlichkeit wurden die Kanten mit unendlicher Kapazität grau bzw. hellorange gezeichnet.

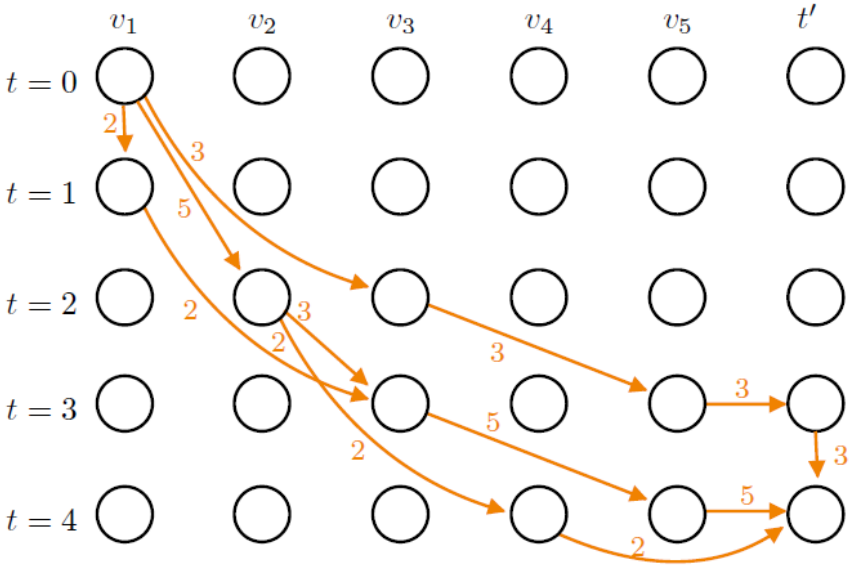
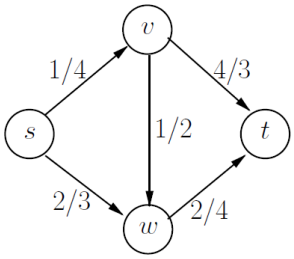


Bild 4.14: Ein möglicher maximaler Fluss des in Bild 4.13 gezeigten Graphen.

Die Abbildung des dynamischen maximalen Flussproblems auf ein statisches Flussproblem durch Vervielfachung der Knoten entsprechend der Zeitebenen stammt von Ford & Fulkerson (1958).

Aufgabe 4.18: Maximales Flussproblem umwandeln

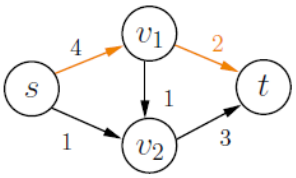
Überführen Sie das nebenstehende dynamische Maximalfluss-Problem (mit Startknoten s und Zielknoten t) in ein Maximalfluss-Problem mit $T = 5$ und lösen Sie dieses Problem. Zur Vereinfachung können Sie davon ausgehen, dass bei den inneren Knoten v und w keine Ware liegen bleiben darf – d.h. dort können die senkrechten Pfeile vernachlässigt werden.



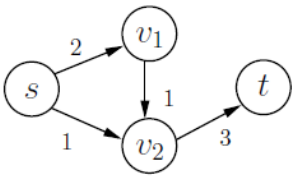
5.5 Flussproblem: Ford-Fulkerson-Algorithmus

In diesem Abschnitt wollen wir einen ersten effizienten Algorithmus für das vereinfachte Problem des maximalen Flusses (Definition 4.57 auf Seite 107) entwickeln.

Die gierige Herangehensweise ist zunächst denkbar einfach: Wir suchen einen beliebigen Weg mit positiven Kapazitätswerten (> 0) an allen Kanten vom Start- zum Zielknoten, bestimmen die maximal mögliche Menge (sprich: die kleinste verfügbare Kapazität an einer Kante des Wegs), merken uns den Fluss und passen die jetzt noch verfügbaren Kapazitäten im Graphen an. Dies iterieren wir solange, bis wir keinen Weg mehr finden. Der Gesamtfluss ergibt sich aus den Flüssen der einzelnen Iterationen.



Graph mit gewähltem Weg im Ford-Fulkerson-Algorithmus.

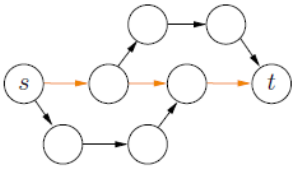


Resultierender (vorläufiger) Restgraph nach dem ersten Schritt.

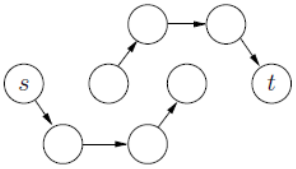
Beispiel 5.30:
Im nebenstehenden Graphen wird der Weg von s über v_1 nach t gewählt. Maximal können 2 Einheiten über diesen Weg geschoben werden. Dadurch verringern sich die für weitere Flüsse noch verfügbaren Kapazitäten an den Kanten: bei (s, v_1) um zwei Einheiten und an der Kante (v_1, t) wird die Kapazität auf 0 reduziert, weswegen wir sie nicht mehr darstellen. In zwei weiteren Iterationen kann noch je eine Einheit über v_2 und über v_1 und v_2 transportiert werden. □

Wir lassen also zunächst einmal etwas vom Start- zum Zielknoten fließen und schauen dann, was jetzt noch fließen kann. Das nenne ich ein gieriges Vorgehen – genau gleich wie beim Shopping: Ich kaufe das erste, was mir gefällt und iteriere, bis das Geld alle ist...

Wenn man ausschließlich wie oben beschrieben vorgeht, kann es passieren, dass der maximale Fluss nicht gefunden wird. Dies illustriert das nächste Beispiel.



Graph mit dem ersten Weg.

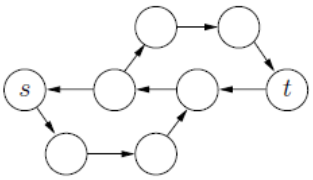


Resultierender (vorläufiger) Restgraph.

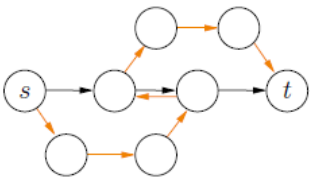
Beispiel 5.31:
Im nebenstehenden Graphen seien alle Kantengewichte $\gamma(e) = 1$, worauf wir aus Gründen der Übersichtlichkeit im Bild verzichten. Es gibt drei Wege vom Startknoten s zum Zielknoten t. Wird hier der mittlere (kürzeste) Weg gewählt, entsteht der nebenstehende Restgraph. Deutlich erkennt man, dass kein weiterer Weg vom Start- zum Zielknoten vorhanden ist. Damit wurde allerdings nicht der maximal mögliche Fluss gefunden – die beiden anderen Wege könnten beide gemeinsam genutzt und damit ein maximaler Fluss von 2 Einheiten erreicht werden. □

Das Beispiel zeigt die Notwendigkeit, dass einmal getroffene Entscheidungen wieder rückgängig gemacht werden können. Konkret: Wurde eine bestimmte Datenmenge über eine Kante transportiert, soll der Rückfluss (eines Teils oder auch der vollständigen Datenmenge) in den weiteren Iterationen möglich sein. Dies wird dadurch erreicht, dass wir für jede Kante des Wegs die verschobene Menge auf die Kapazität der invertierten Kante addieren. War diese Kante vorher nicht vorhanden, wird sie aufgenommen.

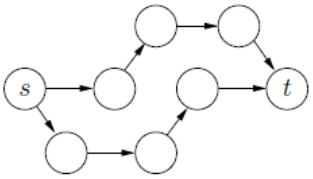
Beispiel 5.32:
Wir setzen Beispiel 5.31 fort. Nebenstehend wird der Graph mit den Rückkanten dargestellt. Die mittlere der Rückkanten ermöglicht dabei jetzt einen weiteren Fluss vom Start- zum Zielknoten. Der zweite nebenstehend dargestellte Graph zeigt beide gefundenen Flüsse. Werden diese Flüsse addiert, heben sich die Flüsse über die mittlere Kante wieder auf und es ergibt sich wie im dritten Bild dargestellt je ein Fluss über den oberen und den unteren Weg. □



Restgraph mit Rückkanten.



Beide gefundenen Flüsse.



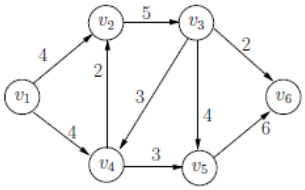
Gesamtfluss.

Gierig entscheiden und dann doch wieder umentscheiden, wenn es dabei etwas zu gewinnen gibt – wenn das doch im Leben auch so wäre :-)

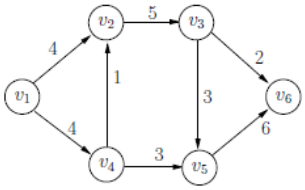
Algorithmus 5.8 (FORD-FULKERSON) enthält schließlich den gesamten Ablauf der gierigen Suche mit Rückkanten für aufgenommene Flüsse. Im zweidimensionalen Feld *fluss* wird der Gesamtfluss für jede Kante aufsummiert, während das Feld *kap* den Restgraphen mit den noch zur Verfügung stehenden Kapazitäten darstellt. In der Variablen *menge* wird für den jeweilig gefundenen Weg der maximale Fluss gespeichert. Diese Variable wird dann verwendet, um die Kantengewichte des Graphen entsprechend anzupassen.

Algorithmus 5.8 Gieriger Algorithmus für das Maximaler-Fluss-Problem

```
FORD-FULKERSON(V, E, γ, s, t)
     Rückgabewert: Fluss pro Kante fluss
 1   E' ← E
 2   for alle Kanten (u, v) ∈ E
 3   do ⌜ kap[u, v] ← γ[u, v] ⌟                          } Restgraph initialisieren
 4        if (v, u) ∉ E
 5        then ⌜ E' ← E' ∪ {(v, u)} ⌟                    } Rückkanten aufnehmen
 6                ⌞ kap[v, u] ← 0
 7        fluss[u, v] ← 0                                } Flussgraph initialisieren
 8      ⌞ fluss[v, u] ← 0
 9   while es existiert Weg (s =)v1, v2, ..., vk(= t) mit   } beliebigen Weg wählen
              (vi, vi+1) ∈ E' und kap[vi, vi+1] > 0 (1 ≤ i ≤ k − 1)
10   do ⌜ menge ← min{kap[vi, vi+1] | 1 ≤ i ≤ k − 1}
11        for i ← 1, ..., k − 1
12        do ⌜ if fluss[vi+1, vi] > 0                    } Flussgraph aktualisieren
13             then ⌜ if fluss[vi+1, vi] ≥ menge
14                  then ⌊ fluss[vi+1, vi] ← fluss[vi+1, vi] − menge
15                  else ⌜ fluss[vi, vi+1] ← menge − fluss[vi+1, vi]
16                     ⌞      ⌞ fluss[vi+1, vi] ← 0
17             else ⌊ fluss[vi, vi+1] ← fluss[vi, vi+1] + menge
18             kap[vi, vi+1] ← kap[vi, vi+1] − menge   } Restgraph aktualisieren
19      ⌞      ⌞ kap[vi+1, vi] ← kap[vi+1, vi] + menge
20   return fluss
```



Ausgangsgraph.



Resultierender Flussgraph.

Beispiel 5.33:

Wir wenden Algorithmus 5.8 auf den nebenstehenden Graphen an. Die einzelnen Iterationen sind in Bild 5.10 dargestellt. Die jeweiligen Flüsse der Iterationen sind dabei variabel, da der Algorithmus FORD-FULKERSON keine Vorgaben bezüglich der Wahl der Wege im Graphen macht. Es ergibt sich der nebenstehende Gesamtfluss. □

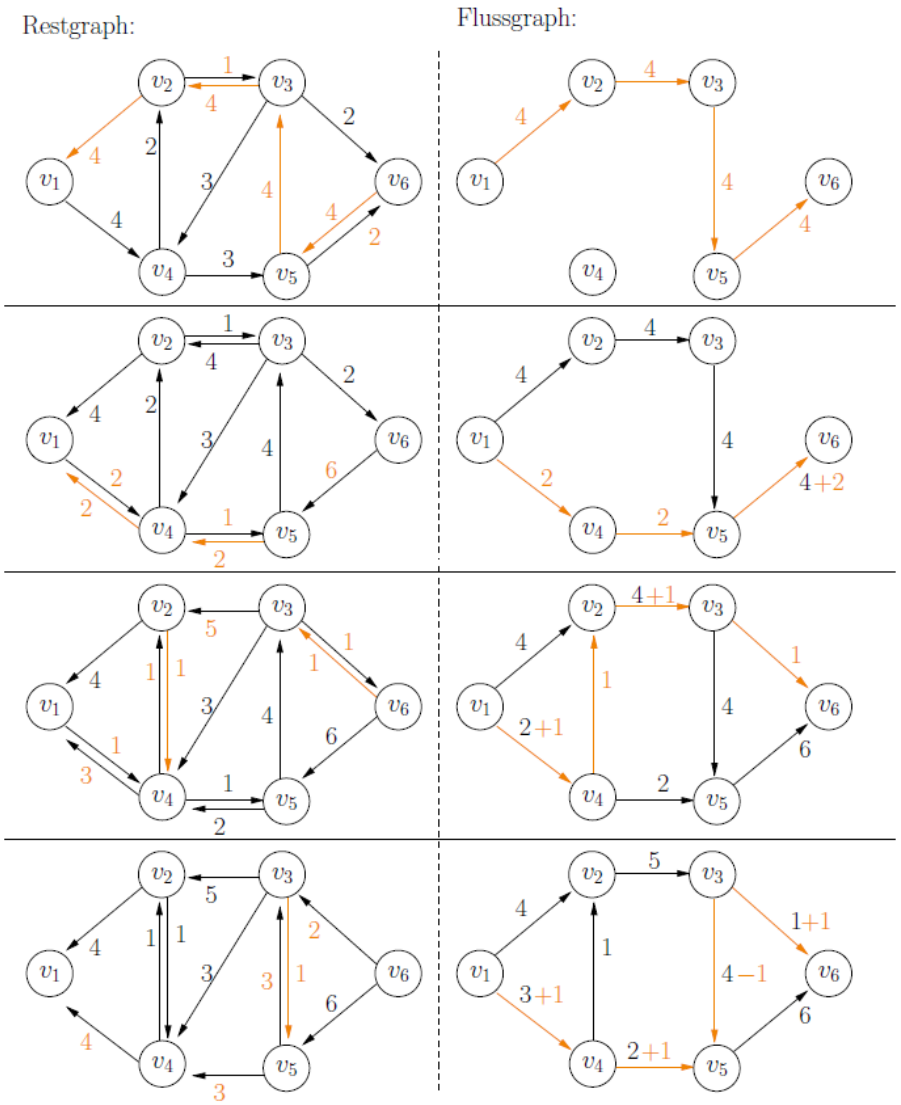


Bild 5.10: In vier Iterationen wird der Gesamtfluss in Beispiel 5.33 aufgebaut. Im rechts angeführten Flussgraph ist der Fluss der jeweiligen Iteration orange markiert. Der resultierende Restgraph ist jeweils links notiert, wobei hier die aktuell neuen Rückkanten orange dargestellt sind.

Satz 5.34: Termination und Korrektheit

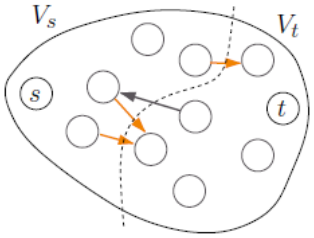
FORD-FULKERSON *(Algorithmus 5.8) terminiert und berechnet den maximalen Fluss.*

Beweis 5.35:

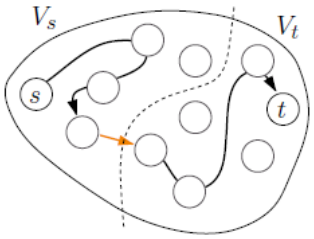
Für den Beweis betrachtet man sog. Schnitte, mit denen die Menge der Knoten in zwei disjunkte Teilmengen $V_s, V_t \subset V$ getrennt wird ($V_s \cup V_t = V$), wobei wir sicherstellen, dass für den Startknoten $s \in V_s$ und für den Zielknoten $t \in V_t$ ist. Dann gilt offensichtlich, dass für einen beliebigen Schnitt die Kanten von V_s nach V_t mit ihren summierten Kapazitäten den maximal möglichen Fluss nach oben beschränken – nebenstehend orange dargestellte Kanten. Von allen möglichen Schnitten bestimmt derjenige mit der kleinsten Gesamtkapazität den maximalen Fluss am Ende der Optimierung. Im Weiteren betrachten wir diesen Schnitt durch den Graphen.

Da jede Iteration durch den Algorithmus einen Weg der Kapazität ≥ 1 zum Gesamtfluss hinzufügt, wächst der Fluss beständig an, und da die Kapazität an der Kante von V_s nach V_t (im beispielhaften Bild orange dargestellt) verringert wird, terminiert der Algorithmus spätestens, wenn die den Schnitt überquerenden Kanten »aufgebraucht« sind.

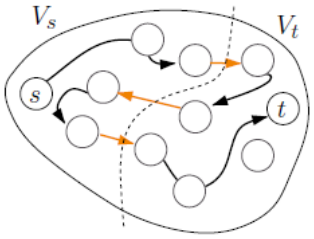
Jetzt müssen wir für die Korrektheit nur noch zeigen, dass der Algorithmus nicht vor dem maximalen Fluss terminiert. Dieser Fall könnte höchstens dann eintreten, wenn ein Fluss in einer Iteration einen Schnitt mehrfach überquert und dadurch »zu viel« Kapazität am Schnitt verbraucht (da die Kapazität ja nur einfach am Zielknoten ankommt). Doch wie die nebenstehende Abbildung zeigt, muss dann der Schnitt in beide Richtungen überquert werden – und die Überquerung von V_t nach V_s erhöht bei der Anpassung der Kantenkapazitäten die Kapazität der entsprechenden Rückkante, die von V_s nach V_t führt. So wird effektiv der Fluss entlang des Wegs nur einmal am Schnitt wirksam – und jede Kapazitätseinheit, die am Schnitt abgezogen wird, kommt als Fluss am Zielknoten an. ■



Schnitt durch den Graph.



Schnitt durch den Graph mit beispielhaftem Weg.



Weg, der einen Schnitt mehrfach überquert.

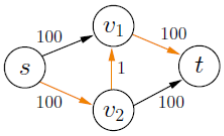
Die Korrektheit des Algorithmus bedeutet hier, dass immer ein Flussgraph gefunden wird, der dem maximal möglichen Fluss entspricht. Da wir in Zeile 9 des Algorithmus die Wahl des nächsten Weges offen halten, können je nach gewählten Wegen unterschiedliche Flussgraphen entstehen.

Satz 5.36: Laufzeit

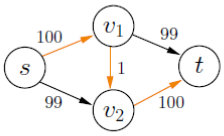
Bezeichne $f_{\max}$ den maximalen Fluss eines Flussproblems mit dem Graphen $G = (V, E)$. Die Laufzeit von FORD-FULKERSON *(Algorithmus 5.8) ist nach oben durch $O(f_{\max} \cdot \#E)$ beschränkt (bei geeigneter Speicherung des Graphen z.B. als Adjazenzliste).*

Beweis 5.37:

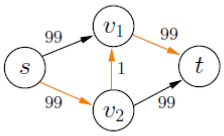
In jeder Iteration muss ein Weg gesucht werden. Im Falle der Speicherung in einer Adjazenzliste benötigt sowohl eine Tiefen- als auch eine Breiten-



Ausgangsgraph für den Laufzeitbeweis.



Restgraph nach einer Iteration.



Restgraph nach der zweiten Iteration.

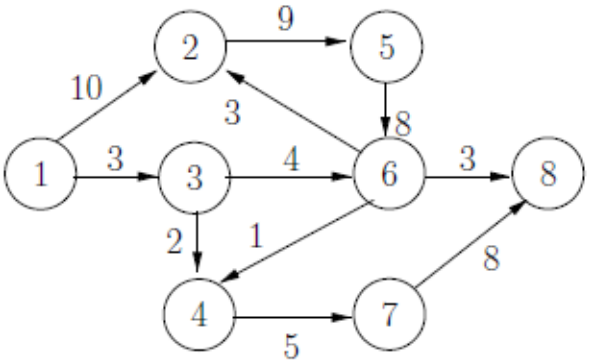
suche $\Theta(\#V + \#E)$. Da wir von einem zusammenhängenden Graphen ausgehen, ist dies gleich $\Theta(\#E)$. Auch die Aktualisierung des Graphen entlang des Wegs benötigt maximal diesen Aufwand. Daher bleibt die Frage wie oft der Algorithmus iterieren kann, wozu wir das nebenstehende Beispiel betrachten, in dem wir den orangefarbigen Weg wählen. Dadurch ergibt sich der zweite Graph als Restgraph. Wird wieder ein Weg über die senkrechte Kante gewählt, ergibt sich der dritte Graph als Restgraph. Wie man sofort erkennt, kann dies maximal $f_{\max}$ oft geschehen. Daraus resultiert die angegebene Gesamtlaufzeit. ■

Diese Laufzeit ist sehr unangenehm, da sie nicht ausschließlich von der Größe des Graphen, d.h. der Anzahl seiner Knoten und Kanten, abhängt.

Der Algorithmus FORD-FULKERSON wurde von Ford & Fulkerson (1956) präsentiert.

Aufgabe 5.7: Maximaler Fluss

Wenden Sie FORD-FULKERSON (Algorithmus 5.8) auf das Flussproblem mit Startknoten 1 und Zielknoten 8 an.



6.1 Verbesserung für den maximalen Fluss

Im letzten Kapitel wurde gezeigt, dass der Algorithmus FORD-FULKERSON (Algorithmus 5.8) in seiner Laufzeit vom maximalen Fluss abhängt. Das bedeutet, dass derselbe Graph durch Verzehnfachung einiger Kantengewichte auf die zehnfache Laufzeit kommen kann. Dies liegt daran, dass immer im Wechsel ein Weg über eine Kante und ihre Rückkante gewählt werden kann (vgl. Beweis zu Satz 5.36). Dieses Ping-Pong lässt sich einschränken, indem die Wege in einer speziellen Reihenfolge betrachtet werden. Konkret werden wir im Weiteren die BREITENSUCHE (Algorithmus 4.23) in Zeile 9 von Algorithmus 5.8 benutzen und so immer den kürzesten Weg vom Start- zum Zielknoten zur Bestimmung des Flusses wählen. Der resultierende Algorithmus wird auch als Edmonds-Karp-Algorithmus bezeichnet.

Beispiel 6.1:
Wir wenden den Edmonds-Karp-Algorithmus auf den Graphen aus Beispiel 5.33 an. Der Ablauf wird Bild 6.1 dargestellt. Die Anzahl der Iterationen bleibt in diesem Beispiel jedoch gleich. □

Satz 6.2: Laufzeit
Die Laufzeit des Edmonds-Karp-Algorithmus ist in $O(\#V \cdot (\#E)^2)$ *(bei geeigneter Speicherung des Graphen z.B. als Adjazenzliste).*

Beweis 6.3:
Die Auswahl des jeweils kürzesten Wegs für den Fluss der nächsten Iteration bewirkt, dass die Länge des Wegs vom Startknoten zu allen anderen

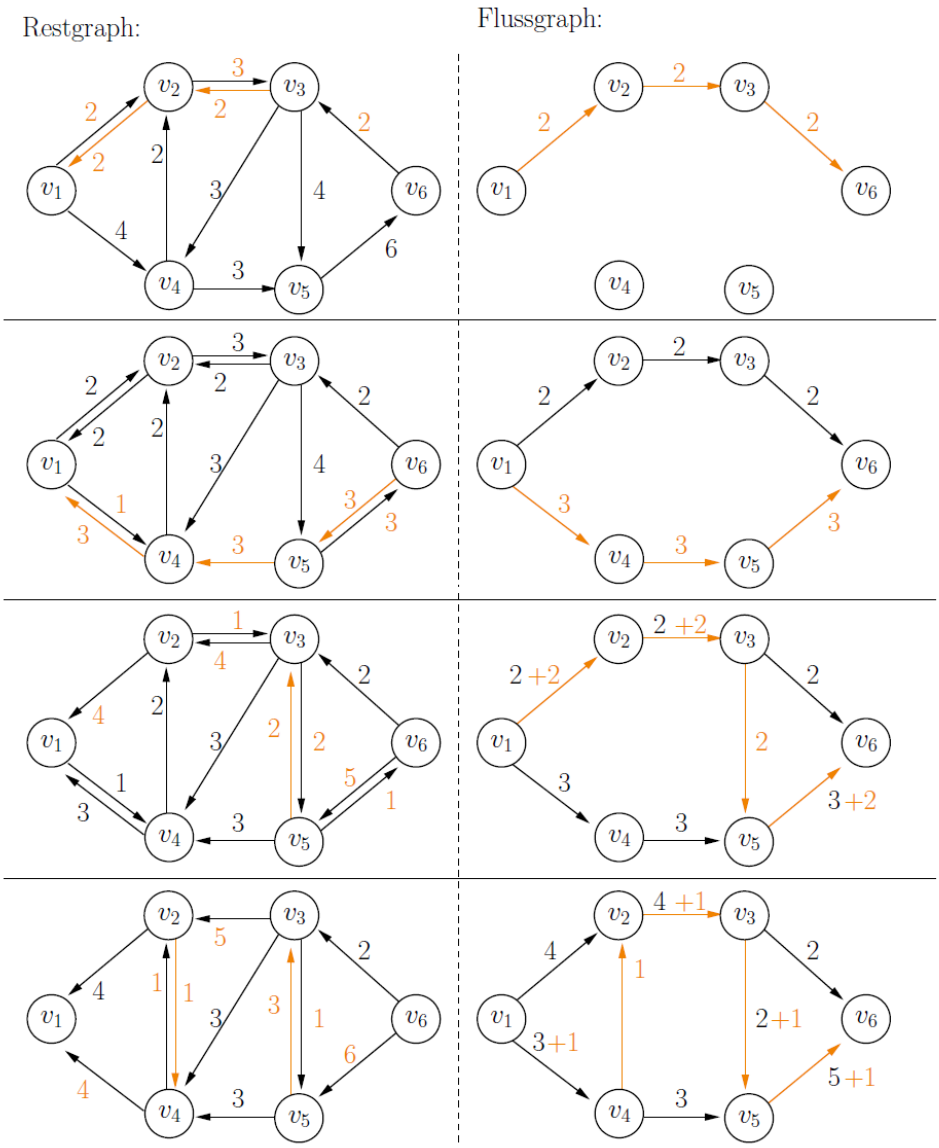
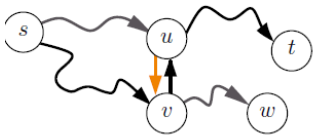


Bild 6.1: In vier Iterationen wird der Gesamtfluss in Beispiel 6.1 mit dem Edmonds-Karp-Algorithmus aufgebaut. Im rechts angeführten Flussgraph ist der Fluss der jeweiligen Iteration orange markiert. Der resultierende Restgraph ist jeweils links notiert, wobei hier die aktuell neuen Rückkanten orange dargestellt sind.

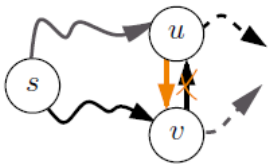


Beweis der Monotonie der Weglängen.

Knoten nie kürzer werden. Dies sieht man leicht durch den nebenstehend skizzierten Widerspruchsbeweis: Angenommen es existiere ein Knoten w, der nach einem Flussschub einen kürzeren Weg hat als zuvor. Dieser muss durch eine neue (orange dargestellte) Rückkante (u, v) entstanden sein – der letzte Fluss ist schwarz dargestellt. Damit ist der graue Weg von s nach u in jedem Fall kürzer als der schwarze Fluss von s nach v (sonst wäre der schwarze Weg von s nach v und dann weiter nach w kürzer gewesen). Dann

kann aber auch der schwarze Weg nicht der kürzeste gewesen sein, da mit dem grauen Weg von s nach u und dann weiter nach t ein kürzerer vorliegt. Daraus folgt, dass es keinen solchen Knoten w geben kann. Also: Bei Verwendung der Breitensuche werden die kürzesten Wege im Graphen länger (oder bleiben gleich lang).

Dieses Ergebnis benutzen wir im Weiteren, wenn wir uns anschauen, wie oft eine Kante verschwinden (d.h. Kapazität 0) und wieder erscheinen kann. Betrachten wir nebenstehendes Beispiel. Der schwarze Weg sei wieder der kürzeste im Graphen und die Kante (v, u) verschwinde im Restgraphen. Dann muss wieder der graue Weg nach u, der dann der kürzeste Weg nach u im Graphen ist, mindestens um eine Kante länger als der schwarze Weg von s nach v sein. Damit (v, u) wieder erscheint, muss (u, v) auf dem kürzesten Weg von s nach t liegen. Dann ergibt sich aber wegen obiger Monotonie der Weglängen (und mit derselben Argumentation wie soeben), dass die jetzt kürzeste Entfernung nach v um 2 größer ist, als vor dem Verschwinden der Kante. Da der längste Weg im Graphen höchstens $\#V - 1$ Kanten lang sein kann, kann maximal $\frac{\#V-1}{2}$ oft eine Kante verschwinden und wieder erscheinen. Da bei jeder Iteration mindestens eine Kante verschwindet, ist die Anzahl der Iterationen in $O(\#V \cdot \#E)$. Mit der Laufzeit der Breitensuche folgt der Satz direkt. ■



Anzahl der Iterationen.

Dieser Satz bedeutet also, dass die Laufzeit nur von der Größe des Graphen, sprich: der Anzahl der Knoten und Kanten, abhängt. Es kann also nicht passieren, dass die Laufzeit aufgrund einer Vervielfachung von Kantengewichten immer weiter wächst, was beim Algorithmus Ford-Fulkerson noch der Fall war.

Beim Ford-Fulkerson-Algorithmus haben wir zufällig und gierig entschieden und dabei jederzeit die Möglichkeit von Rückentscheidungen offen gehalten. Dadurch konnte uns die Laufzeit in Abhängigkeit vom maximal möglichen Fluss aus dem Ruder laufen. Wenn wir jedoch wie beim Edmonds-Karp-Algorithmus ein Schema über die Wahl der Wege vorgeben, können wir die Laufzeit deutlich besser in den Griff bekommen.

Da der Edmonds-Karp-Algorithmus lediglich im Algorithmus Ford-Fulkerson die Breitensuche als Mittel der Wegfindung vorschreibt, wurde dies schon bald nach der Einführung des Ford-Fulkerson-Algorithmus zum Teil so gehandhabt. Edmonds & Karp (1972) gelang schließlich der Beweis, dass diese Vorgehensweise zu polynomieller Laufzeit (in der Größe des Graphen) führt. Inzwischen sind wesentlich schnellere Algorithmen für das Problem des maximalen Flusses bekannt: Die Push/Relabel-Methode von Goldberg & Tarjan (1988a) erreicht eine Laufzeit von $O((\#V)^3)$, was allerdings noch nicht der Algorithmus mit der bestmöglichen asymptotischen Laufzeit ist.

13.5 Maximaler Fluss

Für das Problem des maximalen Flusses wurden lediglich zwei Varianten desselben Algorithmus vorgestellt, die in Tabelle 13.5 verglichen werden. Es wird jeweils eine Speicherung des Graphen als Adjazenzliste vorausgesetzt.

Tabelle 13.5: Vergleich der Laufzeiten der Maximaler-Fluss-Algorithmen

Algorithmus	Laufzeit	Anmerkung
Ford-Fulkerson	$O(f_{\max} \cdot \#E)$	problematisch ist der Einfluss des maximalen Flusses auf die Laufzeit
Edmonds-Karp	$O(\#V \cdot (\#E)^2)$	

Für die zunächst betrachteten Flussprobleme unter Berücksichtigung der Übermittlungsdauer/Reisezeit pro Kante sind die Laufzeiten der obigen Algorithmen allerdings kaum akzeptabel, da bei der Umwandlung des Problems allein schon die Anzahl der Knoten auf $t_{\max} \cdot \#V$ anwächst (mit dem Zeithorizont $t_{\max}$).

Es gibt eine Vielzahl an wesentlich schnelleren Algorithmen. So erreichen Varianten der Push-Relabel-Algorithmen Laufzeiten wie $\Theta((\#V)^3)$.

Die Push-Relabel-Algorithmen wurden von Goldberg & Tarjan (1988b) eingeführt, wobei insbesondere die Heuristiken nach (Cherkassky & Goldberg, 1997) für die Laufzeit hilfreich sind.