4.5.3 Elementarer Ford-Fulkerson-Algorithmus

Sei $G = (V, E)$ ein Netz mit den Kapazitäten $c : E \to \mathbb{R}_{\geq 0}$ und s, t zwei Knoten von G.

1. initialisiere f mit 0
2. solange ein augmentierender Weg P von s nach t in G_f existiert
3. für jede Kante e auf P erhöhe den Fluss um $c_f(P)$

Algorithm 8: Ford-Fulkerson-Algorithmus (FFA)

Um in der zweiten Zeile des Algorithmus einen augmentierenden Weg zu finden benötigt man zwei Schritte:

2a. Konstruktion bzw. Aktualisierung des Restnetzes G_f

2b. Finden eines augmentierenden Weges

Dazu kann man als Datenstruktur den gerichteten Graphen $G' = (V, E')$ mit den Kanten $E' = \{(u, v) \,|\, (u, v) \text{ oder } (v, u) \in E\}$ benutzen. Jedes Restnetzwerk G_f ist ein Teilgraph von G'.

Ein Beispiel für das Vorgehen des Ford-Fulkerson Algorithmus zeigt Abbildung 4.19.

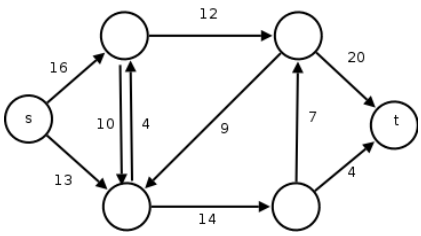
Analyse der Laufzeit:
Schritt 1: $O(|E|)$
Schritt 2a: $O(|E|)$ pro Durchlauf
Schritt 2b: $O(|E|)$ pro Durchlauf, wenn z.B. Tiefen- oder Breitensuche benutzt wird.

Frage 4.5.7. *Wie viele Durchläufe gibt es?*

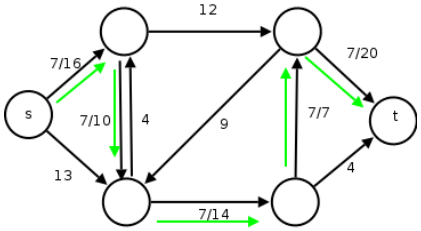
Nehmen wir an, dass die Kapaztitäten ganze Zahlen sind. Dann erhöht jeder Durchlauf den Fluss um mindestens eins, also gibt es bis zu $|f^*|$ Durchläufe, wobei f^* der maximale Fluss ist. Also ist die Laufzeit insgesamt $O(|f^*| \cdot |E|)$. Das gleiche Argument funktioniert, wenn die Kapazitäten rationale Zahlen sind, indem man zunächst mit dem kleinsten gemeinsamen Nenner multipliziert, den Algorithmus laufen lässt und dann wieder durch den Nenner teilt.

Die Laufzeit $O(|f^*| \cdot |E|)$ ist nicht befriedigend, da f^* eventuell exponentiell in der Größe der Eingabe ist. Außerdem gilt die Laufzeitanalyse nur für ganze Zahlen, und es existiert ein Beispiel, wo tatsächlich $|f^*|$ Durchläufe ausgeführt werden. Ein solches Beispiel zeigt Abbildung 4.20. Bei diesem Beispiel werden die augementierenden Wege mit Tiefensuche gefunden. In jedem Schritt wird der Fluss um 1 erhöht und die in (b) und (c) dargestellten Schritte werden $|f^*| = 2000$ mal wiederholt bis der maximale Fluss erreicht ist.

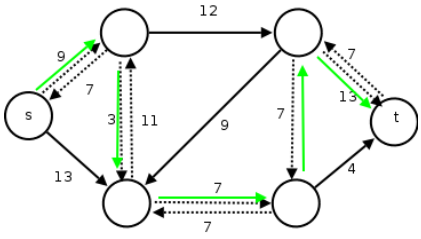
Wenn bei demselben Beispiel die augementierenden Weg mit Breitensuche bestimmt werden, dann werden nur zwei Durchläufe benötigt, wie in Abbildungen 4.21 dargestellt.



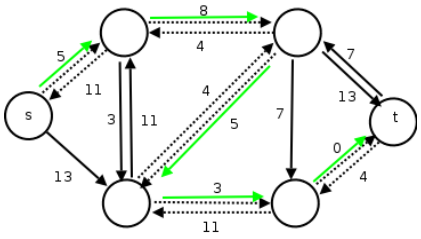
(a) Graph G



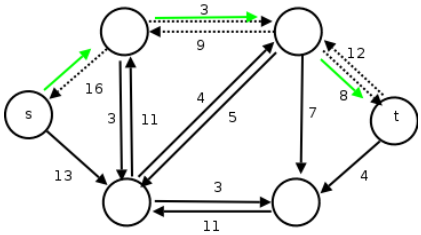
(b) augmentierender Weg mit Kapazität 7



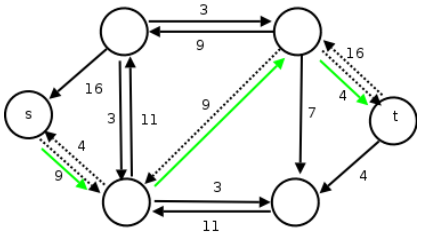
(c) Restnetz nach Schritt 1, augmentierender Weg mit Kapazität 4



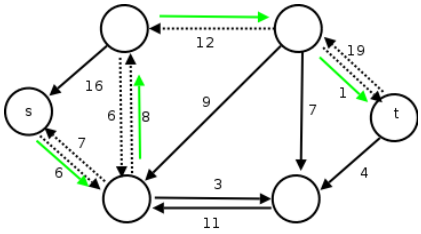
(d) Restnetz nach Schritt 2, augmentierender Weg mit Kapazität 5



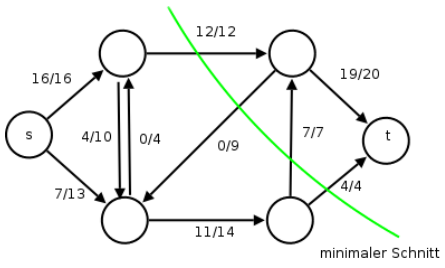
(e) Restnetz nach Schritt 3, augmentierender Weg mit Kapazität 4



(f) Restnetz nach Schritt 4, augmentierender Weg mit Kapazität 3

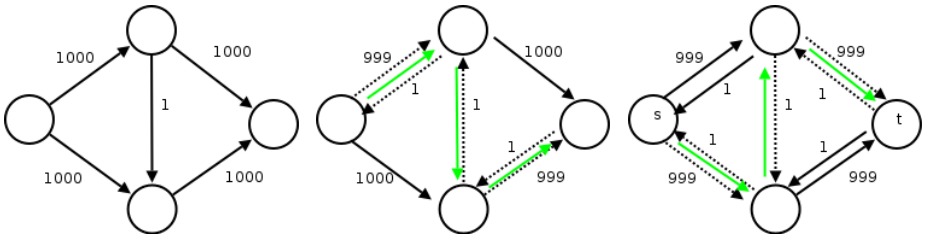


(g) Restnetz nach Schritt 5, kein augmentierender Weg



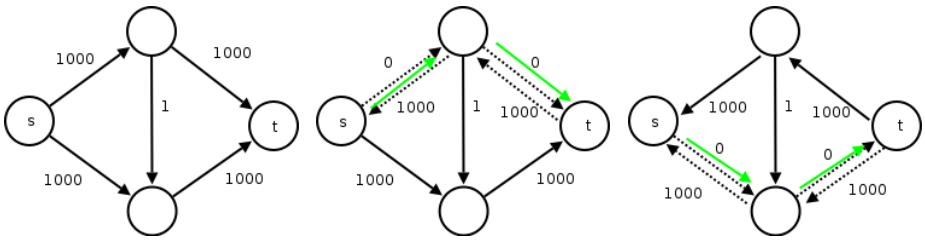
(h) Maximaler Fluss und minimaler Schnitt, beide mit Wert 23

Abbildung 4.19: Beispiel für den Ford-Fulkerson-Algorithmus. Augmentierende Wege und der minimale Schnitt sind in grün eingezeichnet, Kanten im Restnetz deren Kapazität sich geändert hat sind geschrichelt.



(a) Netz (b) Schritt 1 (c) Schritt 2

Abbildung 4.20: Bestimmen des augementierenden Weges mit Tiefensuche.



(a) Netz (b) Schritt 1 (c) Schritt 2

Abbildung 4.21: Bestimmen des augementierenden Weges mit Breitensuche.

4.5.4 Edmonds-Karp-Algorithmus

Es ist ersichtlich aus den Bespielen, dass die Breitensuche Vorteile hat gegenüber der Tiefensuche. Ein Algorithmus der dies nutzt, ist der Algorithmus von Edmonds und Karp.

1. initialisiere f mit 0
2. solange ein augmentierender Weg P von s nach t in G_f existiert
   - 2a. Konstruktion bzw. Aktualisierung des Restnetzes G_f
   - 2b. Finden des augmentierenden Weges mit Breitensuche
3. für jede Kante e auf P erhöhe den Fluss um $c_f(p)$

Algorithm 9: Edmonds-Karp-Algorithmus

Bei diesem Algorithmus wird der kürzeste augmentierende Weg bezüglich der Kantenzahl ausgewählt. Sei $\delta_f(u, v)$ der Abstand zwischen u und v im Restnetz, also die Anzahl der Kanten auf dem kürzesten Weg von u nach v. Dann gilt:

Lemma 4.5.8. *Beim Edmonds-Karp-Algorihtmus gilt für alle Knoten $v \in V\backslash\{s, t\}$: Während des Ablaufs des Algorithmus ist $\delta_f(s, v)$ monoton wachsend.*

Beweis. Angenommen $\delta_f(s,v)$ wächst nicht. Dann exisitiert ein Knoten $v \in V$, ein Fluss f und ein Fluss f' im nächsten Schritt des Algorithmus, so dass $\delta_{f'}(s,v) < \delta_f(s,v)$. Sei v der Knoten mit dieser Eigenschaft, für den $\delta_{f'}(s,v)$ minimal ist. Also gilt für alle anderen Knoten u: $\delta_{f'}(s,u) < \delta_{f'}(s,v) \Rightarrow \delta_f(s,u) \leq \delta_{f'}(s,u)$ $(*)$. Sei P' der kürzeste Weg von s nach v in $G_{f'}$ und u der letzte Knoten vor v auf P'. Dann gilt $\delta_f(s,u) \leq \delta_{f'}(s,u)$ $(**)$ nach $(*)$. Betrachten wir $f(u,v)$, den Fluss von u nach v vor der Augmentierung.

Fall a) $f(u,v) < c(u,v)$. Dann ist (u,v) eine Kante in G_f und es gilt

$$\begin{aligned} \delta_f(s,v) &\leq \delta_f(s,u) + 1 \\ &\leq \delta_{f'}(s,u) + 1 \\ &= \delta_{f'}(s,v) \quad \text{Widerspruch zur Annahme.} \end{aligned}$$

Fall b) $f(u,v) = c(u,v)$. Dann ist (u,v) keine Kante in G_f, aber in $G_{f'}$. Daraus folgt, dass der augmentierende Weg P die Kante (v,u) enthalten haben muss und es gilt

$$\begin{aligned} \delta_f(s,v) &= \delta_f(s,u) - 1 \\ &\leq \delta_{f'}(s,u) - 1 \quad \text{nach } (**) \\ &= \delta_{f'}(s,v) - 2 \\ &< \delta_{f'}(s,v) \quad \text{Widerspruch zur Annahme.} \end{aligned}$$

□