Algorithmik I

Mitschrift

10. März 2011

Inhaltsverzeichnis

1 Elementares über Graphen

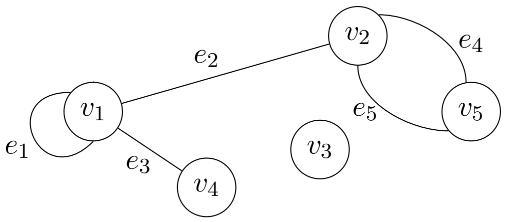
Definition 1.1 Ein ungerichteter Graph ist ein Paar $G = (V, E)$, mit V, E Mengen. V heißt Menge von Knoten, E heißt Menge von Kanten. Außerdem gibt es eine Funktion $i : E \to \mathcal{P}(V)$ mit $\mathcal{P}(V) = 2^V$ (Potenzmenge) und $0 < i(e) \leq 2$. i gibt die Endpunkte einer Kante an.

Ist $i(e) = \{u, v\}$, so heißen u, v Endpunkte von e. Ist $i(e_1) = i(e_2)$, so heißen e_1, e_2 parallel. Ist $|i(e)| = 1$, so heißt e Schleife.

Der Grad eines Knotens v, $grad(v)$, ist die Anzahl der Kanten, für die v Endpunkt ist, wobei Schleifen doppelt gezählt werden. Ist $grad(v) = 0$, so heißt v isoliert.

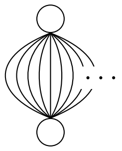
Ein Graph heißt endlich, wenn V und E endlich sind.

Beispiel



v_3	isoliert
e_4, e_5	parallel
e_1	Schleife
$i(e_4) =$	$\{v_2, v_5\} = i(e_5)$
$grad(v_1) =$	4
$grad(v_2) =$	3

Beispiel Unendliche Graphen



unendlich viele Kanten

unendlich viele Knoten

Bemerkung In einem endlichen Graph ist die Anzahl der Knoten mit ungeradem Grad gerade.

Beweis Sei $V = \{v_1, \dots, v_n\}$, dann ist $\sum_{i=1}^{n} grad(v_i) = 2|E|$. Denn: starten wir mit $G = (V, \emptyset)$ und fügen die Kanten nacheinander ein, dann erhöht das Einfügen den Grad beider beteiligten Knoten um jeweils 1. Handelt es sich bei der Kante um eine Schleife, wir der Grad des Knotens um 2 erhöht.

Seien o.E. $v_1 \dots v_j$ mit geradem Grad und $v_{j+1} \dots v_n$ mit ungeradem Grad. $\sum_{l=1}^{j} grad(v_l)$ ist eine gerade Zahl. Über alle Knoten summiert ergibt sich auch eine gerade Zahl $2|E|$.

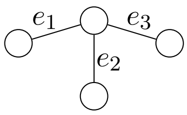
$$2|E| = \sum_{i=1}^{n} grad(v_i) = \underbrace{\sum_{i=1}^{j} grad(v_i)}_{gerade} + \underbrace{\sum_{i=j+1}^{n} \overbrace{grad(v_i)}^{ungerade}}_{gerade}$$

Damit $\sum_{i=j+1}^{n} grad(v_i)$ gerade ist, muss die Anzahl dieser ungeraden Knoten gerade sein.

Definition 1.2 Sei $G = (V, E)$ ein Graph. Sind v_1, v_2 die Endpunkte von e, so heißen v_1, v_2 benachbart. Ein Weg in G ist eine Folge von Kanten $e_1, e_2, \dots$, so dass gilt:

1. $\forall i$ gilt: e_i, e_{i+1} haben einen gemeinsamen Endpunkt.
2. ist e_i keine Schleife und weder erste noch letzte Kante, so hat e_i einen Knoten mit e_{i-1} gemeinsam und den anderen mit e_{i+1}.

Beispiel



Ist e_1, e_2, e_3 ein Weg?
1. ist erfüllt, 2. nicht $\Rightarrow$ kein Weg
e_1, e_2, e_2, e_3 ist ein Weg.

Definition 1.3 Ein Weg wird auch folgendermaßen dargestellt:

$$v_1 \overset{e_1}{\text{——}} v_2 \overset{e_2}{\text{——}} v_2 \overset{e_3}{\text{——}} v_3 \text{——} \dots \overset{e_{n-1}}{\text{——}} v_n$$

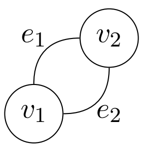
In diesem Weg entspricht e_2 einer Schleife an v_2. Ist dieser abgebildete Weg endlich, so heißen v_1 Anfangspunkt und v_n Endpunkt. Die Länge eines Weges ist gleich der Anzahl der Kanten, die er enthält.

Ein Kreis (Zyklus) ist ein Weg, dessen Anfangspunkt und Endpunkt gleich sind. Ein Weg heißt einfach, wenn jeder Knoten höchstens einmal vorkommt.

Ein Kreis der Länge $\neq 2$ heißt einfach, wenn jeder Knoten außer Anfangs- und Endpunkt höchstens einmal vorkommt.

Ein Kreis der Länge 2 heißt einfach, wenn die beiden Kanten verschieden sind, und wenn jeder Knoten außer dem Start/Endpunkt höchstens einmal vorkommt und der Start/Endpunkt sonst nirgends.

Beispiel



$v_1 \overset{e_1}{\rule{2em}{0.4pt}} v_2 \overset{e_2}{\rule{2em}{0.4pt}} v_1$ einfacher Kreis

$v_1 \overset{e_1}{\rule{2em}{0.4pt}} v_2 \overset{e_1}{\rule{2em}{0.4pt}} v_1$ kein einfacher Kreis

Definition 1.4 Ein Graph $G = (V, E)$ heißt zusammenhängend, wenn es zwischen je zwei Knoten in V einen Weg gibt, der sie verbindet, d.h. einer der Knoten ist Anfangspunkt und einer ist Endpunkt.

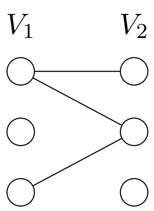
Es gibt für jeden Knoten v der Menge V einen Weg zu v der Länge 0.

Definition 1.5 Sei $G = (V, E)$ ein zusammenhängender Graph. Ein Knoten $a \in V$ heißt Separationspunkt, wenn es Knoten u und v gibt, so dass jeder Weg zwischen u und v über den Knoten a führt.

Hat G einen Separationspunkt, so heißt der Graph separabel, ansonsten unseparabel. Eine Kante e heißt Brücke, wenn es zwei Knoten u, v gibt, so dass jeder Weg von u nach v über die Kante e läuft.

Definition 1.6 Ein Graph $G = (V, E)$ ohne Schleifen heißt bipartit (zweigeteilt), wenn es Mengen $V_1, V_2 \subseteq V$ gibt, so dass für jede Kante gilt: ein Endpunkt liegt in V_1, der andere in V_2.

Beispiel



Definition 1.7 Ein gerichteter Graph ist ein Paar $G = (V, E)$ mit der Menge der Knoten V, der Menge der Kanten E und der Abbildung $i : E \to V \times V$. Ist $i(e) = (u, v)$, so heißt u Anfangspunkt und v Endpunkt von e.

Ist $i(e_1) = i(e_2)$, so heißen e_1 und e_2 parallel. Ist $u \neq v$ und $i(e_1) = (u, v)$ und $i(e_2) = (v, u)$, so heißen e_1, e_2 antiparallel. Ist $i(e) = (v, v)$, so heißt e Schleife.

- $g_{out}(u)$: Anzahl der Kanten, die u als Startpunkt haben. (Ausgrad)
- $g_{in}(u)$: Anzahl der Kanten, die u als Endpunkt haben. (Ingrad)

Bemerkung Für jeden gerichteten Graph mit $V = \{v_1, v_2, \dots, v_n\}$ gilt:

$$\sum_{i=1}^{n} g_{in}(v_i) = \sum_{i=1}^{n} g_{out}(v_i)$$

Begründung: Starte mit dem Graph ohne Kanten. Füge nacheinander alle Kanten ein. Das Einfügen einer Kante e trägt zur Summe auf der linken Seite und zur Summe auf der rechten Seite je 1 bei.

Definition 1.8 Ein gerichteter Weg ist eine Folge von Kanten $e_1, e_2, \dots, e_n$, so dass $\forall i \in \{1, \dots, n-1\}$ gilt: der Endpunkt von e_i ist der Anfangspunkt von e_{i+1}.

Ein endlicher gerichteter Weg heißt Kreis (Zyklus), wenn Anfangspunkt und Endpunkt übereinstimmen. Ein einfacher Weg ist ein Weg, in dem jeder Knoten nur einmal vorkommt.

Ein gerichteter Kreis heißt einfach, wenn kein Knoten außer Anfangs- und Endpunkt mehrfach vorkommt.

Definition 1.9 Ein Graph (gerichtet oder ungerichtet) heißt einfach, wenn er keine parallelen Kanten besitzt.

Definition 1.10 Ein ungerichteter Graph G heißt Kreisfrei, wenn er keinen einfachen Kreis enthält. G heißt Baum, wenn G kreisfrei und zusammenhängend ist.

Definition 1.11 Sei $G = (V, E)$ ein gerichteter Graph. Ein Knoten $v \in V$ heißt Wurzel, wenn von v alle Knoten $w \in V$ aus v erreichbar sind, d.h. $\forall w$ existiert ein gerichteter Weg von v nach w. G heißt gerichteter Baum, wenn G eine Wurzel besitzt und der zugrundeliegende ungerichtete Graph ein Baum ist.

Bemerkung G ist gerichteter Baum genau dann wenn

1. der zugrundeliegende ungerichtete Graph kreisfrei ist,
2. einer der Knoten $r \in V$ die Bedingung $g_{in}(r) = 0$ erfüllt,
3. alle anderen Knoten $v \in V$ die Bedingung $g_{in}(v) = 1$ erfüllen.

Definition 1.12 Sei G ein gerichteter Graph, $v_1, v_2 \in V$. Ein Knoten $v \in V$ heißt *übergeordnet* für v_1, v_2, wenn es einen Weg von v nach v_1 und v_2 gibt.

G heißt übergeordnet, wenn es für je zwei Knoten v_1, v_2 einen Knoten v gibt, der für sie übergeordnet ist.

1. Bäume (endlich oder unendlich) sind übergeordnet.
2. $\bigcirc \leftarrow \bigcirc \leftarrow \bigcirc \leftarrow \ldots$ ist übergeordnet.
3. Jeder Graph mit Wurzel ist übergeordnet.

Bemerkung Jeder endliche übergeordnete Graph hat eine Wurzel.

Definition 1.13 Sei $G = (V, E)$ ein gerichteter Graph. Ein Knoten $v \in V$ heißt Quelle, wenn $g_{in}(v) = 0$, und Senke, wenn $g_{out}(v) = 0$.

2 B-Bäume

Definition 2.1 Ein B-Baum vom Typ t mit $t \geq 2, t \in \mathbb{N}$, ist ein gerichteter Baum T mit Wurzel $T.root$, und hat die folgenden Eigenschaften:

1. Jeder Knoten x enthält die folgenden Informationen:

$$(\underbrace{x.n}_{\text{Anzahl der Schlüssel im Knoten}}, x.c_1, x.key_1, \overbrace{x.c_2}^{\text{Pointer auf Kindknoten}}, \underbrace{x.key_2}_{\text{Schlüssel}} \ldots, x.key_{x.n}, x.c_{x.n+1}, \overbrace{x.leaf}^{\text{Boolean: Knoten ist Blatt}})$$

2. $x.key_i \leq x.key_{i+1} \forall i$ (d.h. gleiche Schlüssel sind erlaubt)
3. Für jeden Schlüssel k_i gilt: steht k_i im Unterbaum mit Wurzel $x.c_i$, so gilt:

$$k_1 \leq x.key_1 \leq k_2 \leq x.key_2 \leq \cdots \leq x.key_{x.n} \leq k_{n+1} \qquad (i = 1 \ldots n+1)$$

4. Alle Blätter haben die gleiche Tiefe.
5. Jeder Knoten außer der Wurzel enthält mindestens $t - 1$ Schlüssel, die Wurzel mindestens 1 Schlüsel.
6. Jeder Knoten enthält höchstens $2t - 1$ Schlüssel. Bei $2t - 1$ Schlüsseln ist der Knoten voll.

Lemma 2.1 Ist $n \geq 1$, so gilt für einen B-Baum T vom Typ $t, t \in \mathbb{N}, t \geq 2$, mit n Schlüsseln: $h(T) \leq \log_t(\frac{n+1}{2})$

Beweis Die Wurzel hat mindestens 1 Schlüssel, die anderen Knoten mindestens $t-1$ Schlüssel. Die Anzahl der Pointer auf Kindknoten ist an die Anzahl der Schlüssel gebunden. Es gibt (sofern der Baum voll genug ist) deshalb mindestens 2 Knoten der Tiefe 1, und mindestens $2t$ Knoten der Tiefe 2, $2t^2$ Knoten der Tiefe 3, usw. In Tiefe n gibt es mindestens $2t^{n-1}$ Knoten.

$$n \geq \overbrace{1}^{\text{Schlüssel in Wurzel}} + (t-1)\overbrace{\sum_{i=1}^{n} 2t^{i-1}}^{\text{andere Schlüssel}} \tag{1}$$

$$n \geq 1 + 2(t-1)\sum_{i=1}^{n} t^{i-1} = 1 + 2(t^n - 1) = 2t^n - 1 \tag{2}$$

$$\frac{n+1}{2} \geq t^n \qquad \text{(log zur Basis } t \text{ anwenden)} \tag{3}$$

$$n \leq \log_t\left(\frac{n+1}{2}\right) \tag{4}$$

Bemerkung Ebenfalls Teil der Vorlesung sind die Algorithmen `B-Tree-Search`, `B-Tree-Create`, `B-Tree-Split-Child`, `B-Tree-Insert` und `B-Tree-Insert-Nonfull`.

3 Flussprobleme

Definition 3.1 Ein *Netzwerk* besteht aus 3 Komponenten:

1. einem endlichen gerichteten Graph $G = (V, E)$ ohne Schleifen (Kanten von v_i zum selben Knoten v_i) und ohne parallele Kanten,
2. einer Funktion c, $c : E \to \mathbb{R}^+$, die jeder Kante $e \in E$ ihre Kapazität zuordnet,
3. 2 ausgezeichneten Knoten s und t, genannt *Quelle* und *Ziel*. Diese Knoten müssen nicht im Graphentheoretischen Sinne Quelle und Senke sein.

Soll ein Graph mit parallelen Kanten in ein Netzwerk überführt werden, ersetzt man alle parallelen Kanten durch jeweils nur eine Kante. Die Kapazität dieser neuen Kante ist gleich der Summe der Kapazitäten der Kanten, die sie ersetzt:

Schleifen $e = (v, v)$ ergeben im Kontext von Flussproblemen keinen Sinn. Falls sie trotzdem in einem Netzwerk dargestellt werden sollen, kann man sie durch Kanten $e' = (v, v')$ und $e'' = (v', v)$ zu einem neuen Knoten v' darstellen.

Definition 3.2 Eine *Flussfunktion* für ein Netzwerk (G, c, s, t) ist eine Funktion $f : E \to \mathbb{R}$, für die gilt:

1. $0 \leq f(e) \leq c(e)$

2. sei $\alpha(v) = \{e : e \text{ führt zu } v\}$, $\beta(v) = \{e : e \text{ führt aus } v\}$, $v \in V$ Knoten, dann gilt $\forall v$ mit $v \neq s$ und $v \neq t$ das *Flusserhaltungsgesetz*:

$$\sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e)$$

Sei f eine Flussfunktion, dann heißt

$$F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e)$$

der *totale Fluss* von f. Der totale Fluss bezieht sich stets auf das – frei wählbare – Ziel des Netzwerks t.

Bemerkung Eine typische Aufgabe ist es, zu einem Netzwerk eine Flussfunktion f zu suchen, so dass ihr totaler Fluss F maximal ist.

Definition 3.3 Sei $S \subseteq V$ mit $s \in S$, $\overline{S} = V \setminus S, t \in \overline{S}$.
$E_{S\overline{S}} = \{e : \text{ Anfangspunkt von } e \in S, \text{ Endpunkt von } e \in \overline{S}\}$
$E_{\overline{S}S} = \{e : \text{ Anfangspunkt von } e \in \overline{S}, \text{ Endpunkt von } e \in S\}$
Der durch S definierte *Schnitt* ist die Menge $E_{S\overline{S}} \cup E_{\overline{S}S}$. Die Kapazität des durch S definierten Schnitts ist

$$c(S) = \sum_{e \in E_{S\overline{S}}} c(e) \qquad \text{(Beachte: dies ist die Kapazität in Richtung } \overline{S},\ t \in \overline{S})$$

Lemma 3.1 Sei $N = (G, c, s, t)$ ein Netzwerk, und f eine Flussfunktion für N. Dann gilt $\forall S \subseteq V$ mit $s \in S$ und $t \notin S$:

$$F = \sum_{e \in E_{S\overline{S}}} f(e) - \sum_{e \in E_{\overline{S}S}} f(e)$$

Egal welches S man wählt, der totale Fluss (gebunden an t) lässt sich damit berechnen.

Beweis

$$0 = \sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e) \qquad \forall v, v \neq s, v \neq t, \text{ weil } f \text{ Flussfunktion} \tag{1}$$

$$F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e) \qquad \text{Definition von } F \tag{2}$$

Addiere Gleichung (1) für alle $v \in \overline{S} \setminus \{t\}$ zu Gleichung (2). Auf der linken Seite der Ergebnisgleichung steht F. Im folgenden bestimmen wir die rechte Seite. Betrachte alle Kanten in E: sei $e \in E$ mit $x \xrightarrow{e} y$.

- Fall 1: $x, y \in S$: dann kommt $f(e)$ in der Summation nicht vor (es wird nur über $v \in \overline{S} \setminus \{t\}$ summiert).
- Fall 2: $x, y \in \overline{S}$: $e \in \alpha(x), e \in \beta(x)$, also heben sich die Werte von $f(e)$ bei der Summation gegenseitig auf.
- Fall 3: $x \in S, y \in \overline{S}$: $f(e)$ ist positiv für y, für x kommt es nicht in der Summation vor, $e \in E_{S\overline{S}}$.
- Fall 4: $x \in \overline{S}, y \in S$: $f(e)$ ist negativ für x, kommt für y nicht in der Summation vor, $e \in E_{\overline{S}S}$.

Bei der Summation über die rechte Seite ergibt sich:

$$\sum_{e \in E_{S\overline{S}}} f(e) - \sum_{e \in E_{\overline{S}S}} f(e) \qquad \text{(Fall 3 - Fall 4)}$$

Beispiel Schreibweise: $^{c}/_{f}$ $\qquad$ $^{\text{Kapazität}}/_{\text{Fluss}}$

Lemma 3.2 Für jede Flussfunktion f mit totalem Fluss F und für jedes $S \subseteq V$ mit $s \in S, t \notin S$ gilt: $F \leq c(S)$.

Beweis Aus Lemma 3.1 folgt:

$$F_{3.1} = \sum_{e \in E_{S\overline{S}}} f(e) - \sum_{e \in E_{\overline{S}S}} \leq \sum_{e \in E_{S\overline{S}}} f(e) \leq \sum_{e \in E_{S\overline{S}}} c(e) \overset{\text{Def}}{=} c(S)$$

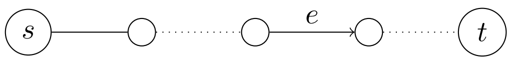
Korollar 3.3 *Max Flow Min Cut-Theorem*: Sei f eine Flussfunktion und $S \subseteq V, s \in S, t \notin S$. Wenn $F = c(S)$, dann ist f eine Flussfunktion mit maximalem totalen Fluss und die Kapazität des durch S definierten Schnittes ist minimal.

Beweis Sei für ein f und ein S der totale Fluss $F = c(S)$. Sei f' eine andere Flussfunktion mit totalem Fluss F'. Aus Lemma 3.2 folgt: $F' \leq c(S) = F$, also ist F maximal.

Sei $S' \subseteq V$ mit $s \in S', t \notin S'$. Dann gilt: $\underset{=c(S)}{F} \leq c(S)$, also ist $c(S)$ minimal.

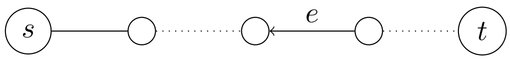
Definition 3.4 Gegeben sei ein Netzwerk (G, c, s, t) und eine Flussfunktion f für dieses Netzwerk. Ein *Anreicherungspfad* (Augmenting Path) ist ein einfacher Pfad oder Weg von s nach t, der nicht notwendigerweise gerichtet ist, und für den gilt: Sei e eine Kante auf diesem Weg:

- Fall 1: e ist eine *Vorwärtskante*, d.h.



  dann muss $f(e) < c(e)$ sein.

- Fall 2: e ist eine *Rückwärtskante*, d.h.



  dann muss $f(e) > 0$ sein.

Definition 3.5 Eine *Vorwärtsmarkierung* des Knoten v durch die Kante $u \xrightarrow{\quad e \quad} v$ ist anwendbar, wenn

1. u markiert ist und v nicht, und
2. $f(e) > 0$.

v erhält dann die Markierung "e" (Name der Kante).

Eine *Rückwärtsmarkierung* des Knoten v durch die Kante $u \xleftarrow{\quad e \quad} v$ ist anwendbar, wenn

1. u markiert ist und v nicht, und
2. $f(e) > 0$.

v erhält dann die Markierung "e".

Im 1. Fall, also bei Vorwärtsmarkierung, wird $\Delta(e) = c(e) - f(e)$. Im 2. Fall wird $\Delta(e) = f(e)$. In beiden Fällen ist $\Delta(e) > 0$.

Definition 3.6 *Algorithmus von Ford und Fulkerson zur Bestimmung einer Flussfunktion mit maximalem totalen Fluss.* Gegeben sei ein Netzwerk (G, c, s, t) und eine Flussfunktion f für dieses Netzwerk. Der folgende Algorithmus verändert f mit dem Ziel, den totalen Fluss zu maximieren.

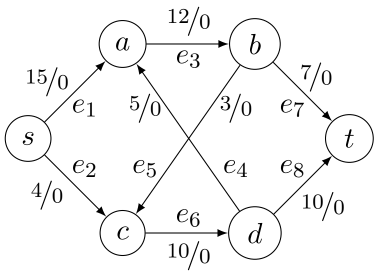
1. Setze $f(e) := 0 \qquad \forall e \in E$
2. Kennzeichne s als "markiert" und alle anderen Knoten als "unmarkiert"
3. Suche einen Knoten v, der vorwärts oder rückwärts markiert werden kann. Gibt es keinen solchen Knoten, dann halte an. Das damit gefundene f hat maximalen totalen Fluss. Gibt es einen solchen Knoten v, dann markiere v mit der Kante e, die zur Markierung geführt hat. Ist $v = t$, dann gehe zu Schritt 4, ansonsten zu Schritt 3.
4. Starte bei t und rekonstruiere mittels der Markierungen den "Weg", auf dem man von s nach t gelangt ist. Dies sei

$$s = v_0 \overset{e_1}{—} v_1 \cdots\cdots\cdots v_{e-1} \overset{e_l}{—} v_l = t$$

Sei $\Delta = min(\{\Delta(e_i) : 1 \leq i \leq l\})$. Ist e_i eine Vorwärtskante, so setze $f(e_i) := f(e_i) + \Delta$, andernfalls $f(e_i) := f(e_i) - \Delta$.

5. Gehe zu Schritt 2.

Beispiel In diesem Beispiel wird der Ford-Fulkerson-Algorithmus auf ein Netzwerk angewendet. Die folgende Abbildung zeigt das Netzwerk vor dem ersten Schritt.



"Wege" in den Iterationsschritten, gefundenes Δ, neue Flusswerte:

1. $$\begin{array}{lccccccccccc} & s & \xrightarrow{e_2} & c & \xrightarrow{e_6} & d & \xrightarrow{e_4} & a & \xrightarrow{e_3} & b & \xrightarrow{e_7} & t \\ \Delta(e): & & 4 & & 10 & & 5 & & 12 & & 7 & \end{array}$$
   $\Rightarrow \Delta = 4$
   $\Rightarrow e_2 \rightarrow 4/4 \quad e_3 \rightarrow 12/4 \quad e_6 \rightarrow 10/4 \quad e_7 \rightarrow 7/4 \quad e_4 \rightarrow 5/4$
   $\Rightarrow F = 4$

2. $$\begin{array}{lccccccccccc} & s & \xrightarrow{e_1} & a & \xrightarrow{e_2} & b & \xrightarrow{e_5} & c & \xrightarrow{e_6} & b & \xrightarrow{e_8} & t \\ \Delta(e): & & 15 & & 8 & & 3 & & 6 & & 10 & \end{array}$$
   $\Rightarrow \Delta = 3$
   $\Rightarrow e_1 \rightarrow 15/3 \quad e_3 \rightarrow 12/7 \quad e_5 \rightarrow 3/3 \quad e_6 \rightarrow 10/7 \quad e_8 \rightarrow 10/3$
   $\Rightarrow F = 7$

3. $$\begin{array}{lccccccc} & s & \xrightarrow{e_1} & a & \xrightarrow{e_3} & b & \xrightarrow{e_7} & t \\ \Delta(e): & & 12 & & 5 & & 3 & \end{array}$$
   $\Rightarrow \Delta = 3$
   $\Rightarrow e_1 \rightarrow 15/6 \quad e_3 \rightarrow 12/10 \quad e_7 \rightarrow 7/7$
   $\Rightarrow F = 10$

4. $$\begin{array}{lccccccc} & s & \xrightarrow{e_1} & a & \xrightarrow{e_4} & d & \xrightarrow{e_8} & t \\ \Delta(e): & & 9 & & 4 & & 7 & \end{array}$$
   $\Rightarrow \Delta = 4$
   $\Rightarrow e_1 \rightarrow 15/10 \quad e_4 \rightarrow 5/0 \quad e_8 \rightarrow 10/7$
   $\Rightarrow F = 14$

5. Nach a und b kann kein weiterer Knoten markiert werden, der Algorithmus bricht ab. f hat jetzt maximalen totalen Fluss.

Lemma 3.4 Nach jedem Durchlauf durch einen der Iterationsschritte des Algorithmus gilt: das aktuell berechnete f ist eine Flussfunktion.

Beweis Trivial für die Schritte 1, 2, 3 und 5. Schritt 4:
Sei f die Flussfunktion vor Eintritt in Schritt 4 und f' die in Schritt 4 berechnete Funktion. f erfülle die Bedingungen

1. $0 \leq f(e) \leq c(e)$
2. $\forall v \in V, v \neq s, v \neq t : \sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e)$

In Schritt 4 wird ein Anreicherungspfad rekonstruiert, dies sei:

$$s = v_0 \overset{e_1}{\text{——}} v_1 \cdots\cdots \overset{e_k}{\text{——}} v_k = t$$

Die Definition von $\Delta(e)$ und Δ garantiert, dass $0 \leq f'(e) \leq c(e)$. Zu zeigen bleibt das Flusserhaltungsgesetz für f'. Wir müssen nur die Knoten anschauen, die auf dem Weg liegen, d.h. $v_1, \ldots, v_{k-1}$. Sei v_i einer dieser Knoten. Es können folgende Fälle auftreten:

- Fall 1: $\xrightarrow[\in \alpha(v_i)]{e_i} v_i \xrightarrow[\in \beta(v_i)]{e_{i+1}}$
  Δ wird zu beiden Kantenflüssen addiert – zu $\sum_{e \in \alpha(v)} f(e)$ und zu $\sum_{e \in \beta(v)} f(e)$.

- Fall 2: $\xrightarrow[\in \alpha(v_i)]{e_i} v_i \xleftarrow[\in \alpha(v_i)]{e_{i+1}}$
  Δ wird zur Summe addiert und (wegen Rückwärtskante) subtrahiert.

- Fall 3: $\xleftarrow[\in \beta(v_i)]{e_i} v_i \xrightarrow[\in \beta(v_i)]{e_{i+1}}$
  Analog zu Fall 2.

- Fall 4: $\xleftarrow[\in \beta(v_i)]{e_i} v_i \xleftarrow[\in \alpha(v_i)]{e_{i+1}}$
  Analog zu Fall 1.

Index