UNIVERSITÄT
LEIPZIG

Algorithmen und Datenstrukturen II

Vorlesung *pe5ë*

Leipzig, 30.04.2024

Peter F. Stadler & Thomas Gatter & Ronny Lorenz

FLÜSSE II

Zur Erinnerung: Ford-Fulkerson Algorithmus

- Graph $G = (V, E)$, mit Flussnetzwerk $F = (G, c, s, t)$. G beschreibt die möglichen Wege für die Flüsse, $c : E \to \mathbb{R}^+$ sind die maximalen Kapazitäten, s und t sind Quelle und Senke.
- Wenn entlang einer Kante ein Fluss > 0 fließt, dann wird auch eine Rückkante mit genau diesem Fluss gesetzt.
  Der maximale Fluss von u nach v ist damit immer durch noch freie Kapazität $u \to v$ plus Fluss $v \to u$ gegeben.
- Für alle Knoten in einem Flussnetzwerk gilt, dass Zu- und Abfluss gleich sind. Ausnahmen sind s und t, diese Knoten sind speziell.

Video, Tim Roughgarden: *(Link)*

Zur Erinnerung

Für Flussnetzwerke gilt:

- *Konservierung*: Bis auf Quelle und Senke muss der Zufluss in jedem Knoten gleich dem Abfluss sein
- *Zulässigkeit*: Der Fluss entlang jeder Kante überschreitet die vorgegebene Kapazitätsgrenze nicht
- Ein maximaler Fluss ist erreicht, wenn es keinen *erweiternden Pfad* (im Restgraphen) mehr von Quelle zu Senke gibt.

Warum Invarianten?

Invarianten helfen, die Korrektheit eines Algorithmus zu beweisen.

- Das Ergebnis zur Aufgabenstellung muss eine Reihe von Bedingungen erfüllen.
- *Hier: Konservierung, Zulässigkeit, Nicht-Existenz eines erlaubten (erweiternden) Pfades*
- Ein Algorithmus wird dann so konstruiert, das **ein Teil** dieser Bedingungen schon am Anfang erfüllt ist, und in jedem Schritt erhalten bleibt.
- Es muss nur mehr sichergestellt werden, dass die restliche(n) Bedindung(en) durch den Algorithmus schliesslich erreicht werden können.

Pre-Flüsse statt Flüsse

Definitionen

- **Pre-Fluss**: Eine Verallgemeinerung des Fluss-Begriffs, so dass der Zufluss in einen Knoten mindestens so groß wie der Abfluss ist.

$$\forall v \in V \setminus \{s,t\} : \sum_{(v',v)\in E} f(v',v) \geq \sum_{(v,v')\in E} f(v,v')$$

- Der **Exzess** $ex(x)$ an einem Knoten, $x \neq s,t$ ist der Unterschied von Zufluss und Abfluss. $ex(x) \geq 0$ in einem Pre-Fluss.

Ein Pre-Fluss f ist ein Fluss, wenn $ex(x) = 0$ für alle $x \in V \setminus \{s,t\}$ gilt.

Push & Relabel - Goldberg-Tarjan 1988 I

Benutze Pre-Flüsse und alternative Invarianten

- **Pre-Fluss-Eigenschaft**: f ist ein Pre-Fluss
- **Zulässigkeit**: Der Pre-Fluss entlang jeder Kante überschreitet die vorgegebene Kapazitätsgrenze nicht
- **Unerreichbarkeit:** Es gibt keinen erlaubten Pfad von Quelle zur Senke im Restgraphen

$\Rightarrow$ Das heißt, wir haben einen maximalen zulässigen Fluss genau dann, wenn f die Konservierungsbedingung erfüllt (wenn der Exzess verschwindet).

Push & Relabel - Goldberg-Tarjan 1988 II

Beginne mit dem Ausfluss an der Quelle s und schiebe den Exzess schrittweise in Richtung Senke t.

Problem

“Buchhaltung” ist nötig, damit immer ein Gefälle gewährleistet ist und Fluss nicht im Kreis verschoben wird.

Höhenfunktion $h(x)$ mit $h(s) = |V|$, $h(t) = 0$, und $h(u) \geq h(v) + 1$ für jede Kante (u, v) entlang der Fluss noch verschoben werden kann. (**Ordnungsbedingung**)

Wie gehabt: Restgraph mit $rest(u, v) = c(u, v) - f(u, v)$ für $e \in E$ und $rest(v, u) = f(u, v)$ für “Rückwärtskanten”

Push & Relabel: Initialisierung

Algorithmus - Initialisiere Graph

- Erzeuge Restgraphen von G
- Setze Höhen $h(s) = |V|$, $h(t) = 0$, $h(v) = 0, v \neq s, t$
- Setze Flüsse $f(s, v) = c(s, v)$ und $f(u, v) = 0, u \neq s$ für die von der Quelle ausgehenden Kanten. An jeder diese Kanten liegt nun ein maximaler Fluss an, alle anderen Kanten haben 0 Fluss
- f ist Pre-Fluss (kein Fluss, da Konservierung verletzt ist). Allerdings gibt es momentan auch keinen erweiternden s-t Pfad mehr (da alle Kanten von s ausgehenden Kanten schon maximal gefüllt sind)

Push & Relabel: Hauptalgorithmus I

Im Beispiel weiter hinten wird zuerst ein Fluss entlang eines Pfades $s - t$ gesetzt, bevor Push-Relabel beginnt. Deshalb sind nicht alle $f(u, v) = 0$. Das stört allerdings den Algorithmus nicht.

Nach der Initialisierung haben einige Knoten x Exzess $ex(x) > 0$. Der Algorithmus läuft nun so lange bis alle Knoten Exzess $ex(x) = 0$ haben. Da immer die Invarianten (Prefluss, Zulässigkeit, Unerreichbarkeit) erhalten bleiben, ist dann ein maximaler Fluss gefunden worden.

Push & Relabel: Hauptalgorithmus II

Algorithmus

(1) Finde Knoten $v \neq s$ mit ex(v)$>$ 0 und dessen erlaubte Kanten.
   - erhöhe h (**relabel**) falls es keine erlaubte Kante gibt

(2) **Push** Fluss von aktivem Knoten über erlaubte Kanten
   - passe $ex(v)$ an (aus Pre-Fluss)
   - passe Restgraph an

– Iteriere (1) und (2) bis $ex(v) = 0$ für alle $v \in V \setminus \{s, t\}$.
  Nach Idee ist dann ein maximaler Fluss gefunden.

Nachfolgend etwas detaillierter.

Push

Aktiver Knoten: beliebiges u mit $ex(u) > 0$

Erlaubte Kante: $rest(u, v) > 0$ und $h(u) = h(v) + 1$.

Details - Push-Schritt

Aus einem aktiven Knoten u, entlang einer erlaubten Kante $e = (u, v)$:

- wenn $e \in E(G)$: $\quad f(e) \leftarrow f(e) + \min\{ex(u), rest(e)\}$
- wenn $e \notin E(G)$: $\quad f(e') \leftarrow f(e') - \min\{ex(u), rest(e)\}$

  e ist im Restgraphen die Rück-Kante zu einer Kante $e' \in E(G)$

Dieser Schritt **pusht** entlang einer *absteigenden* Kante (u, v) mit $h(u) > h(v)$ so viel Fluss wie möglich.

Relabel

Details - Relabel-Schritt

an einem aktiven Knoten u ohne erlaubte Kante:

$$h(u) \leftarrow 1 + \min\{h(v) | (u, v) \in E(G_f)\}$$

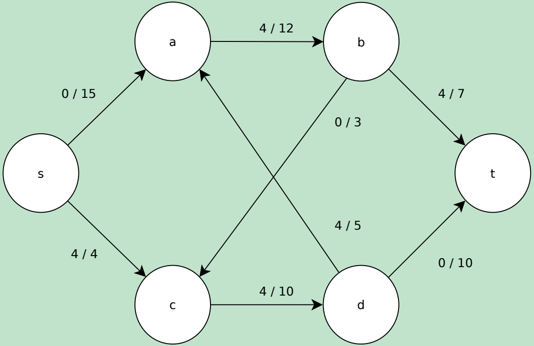
Lassen wir Kanten mit Kapazität 0 in G_f zu, betrachten wir nur v mit $rest(u, v) > 0$!

$\Rightarrow$ Die Kanten zu allen Nachbarn v mit minimaler Höhe werden aktiviert.

Hierbei können Knoten u mit Kanten (s, u) auch eine Höhe erreichen, bei der ein **Push** einen Exzess-Fluss wieder zurück nach s schiebt. Das ist gewollt, da bei diesem (letzten) Schritt für u ein Exzess, der nicht mehr Richtung Senke t fließen kann, entfernt werden muss, um dem Ziel $ex(x) = 0$ für alle Knoten x außer s, t näher zu kommen.

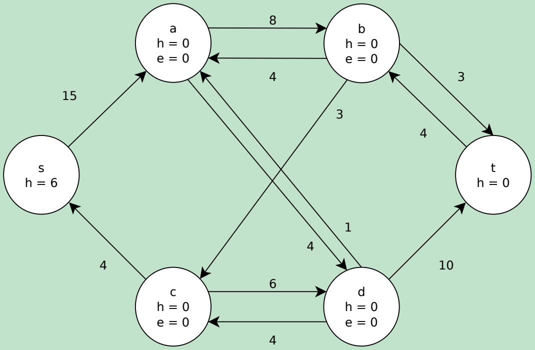
Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



Push and Relabel: Beispiel

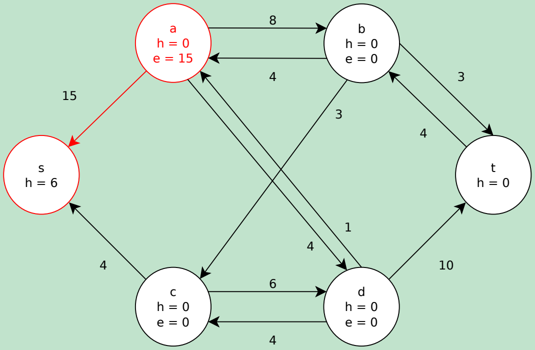
Beispiel (siehe Wikipedia)



[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Beispiel

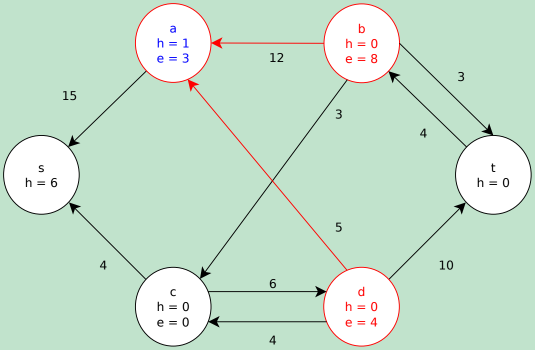
Beispiel (siehe Wikipedia)



[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Beispiel

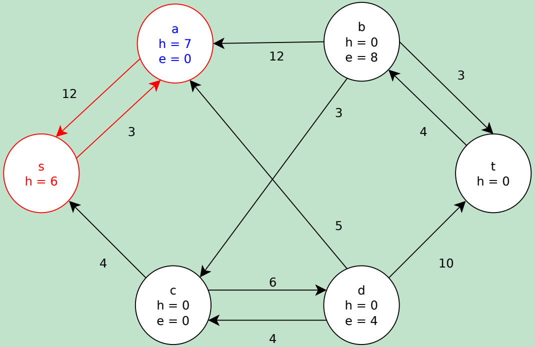
Beispiel (siehe Wikipedia)



[By Kamac124 - Own work, CC BY-SA 4.0]

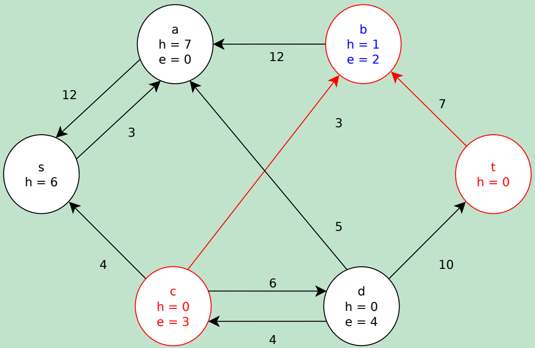
Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



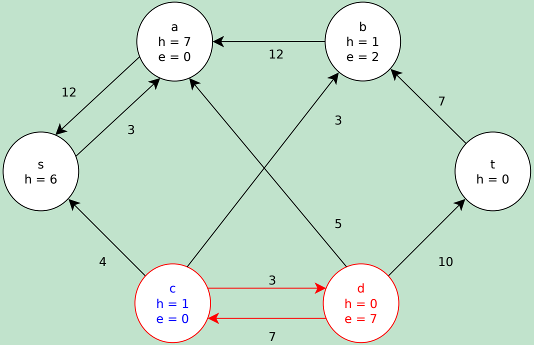
Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



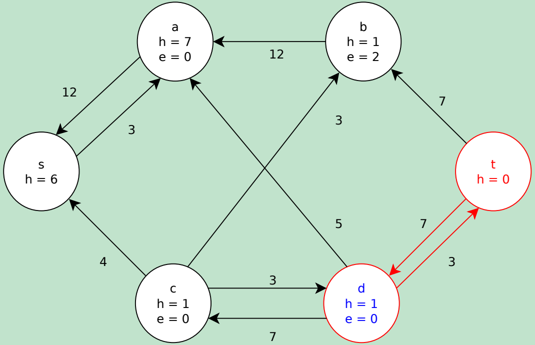
Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



Push and Relabel: Beispiel

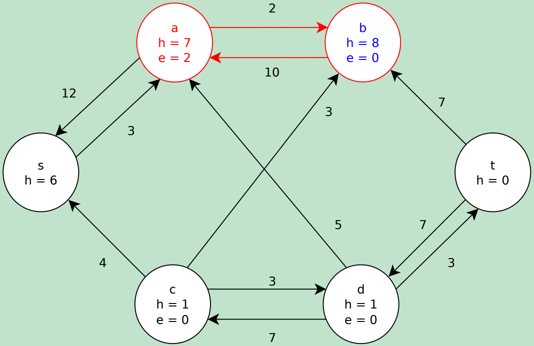
Beispiel (siehe Wikipedia)



[By Kamac124 - Own work, CC BY-SA 4.0]

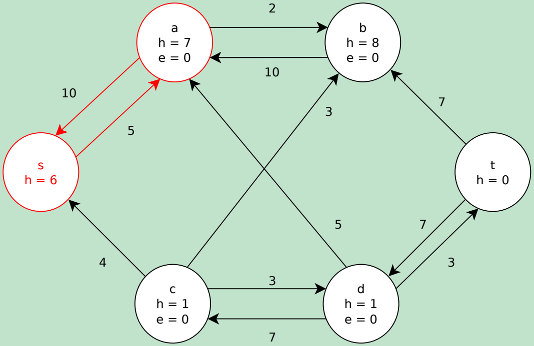
Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



Push and Relabel: Beispiel

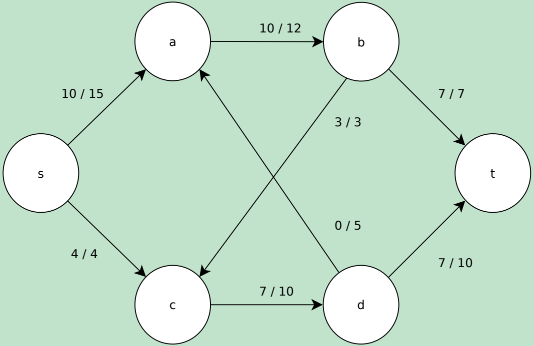
Beispiel (siehe Wikipedia)



[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Beispiel

Beispiel (siehe Wikipedia)



Warum funktioniert das?

- Jeder **Push**-Schritt entlang einer Kante (u, v) erhält die *Pre-Fluss-Eigenschaft*:
  Dabei kann Restkapazität auf der Kante (v, u) erzeugt werden und diese in den Restgraphen aufgenommen werden. Für diese gilt $h(v) \leq h(u) + 1$

- Jeder **Relabel**-Schritt erhält die *Ordnungsbedingung* für h:
  Wenn u keine erlaubte Kante hat, dann gilt $h(u) \leq h(v)$. Nach dem *relabel* Schritt ist daher $h'(u) > h(u)$. Für einlaufende Kanten (p, u) gilt also $h(p) \leq h(u) + 1 < h'(u) + 1$. Für die auslaufenden Kanten gilt $h'(u) \leq h(v) + 1$ per Konstruktion. Da h nur am Knoten u verändert wird, erfüllt h nach dem *Relabel*-Schritt die Ordnungsbedingung

Warum funktioniert das?

- Jeder **push** und jeder **relabel** Schritt erhält die Pre-Fluss und die Ordnungsbedingung.
- Im Restgraphen gibt es hier *keinen* Pfad aus erlaubten Kanten von s nach t: Entlang eines solchen Pfades ändert sich h höchstens um 1 entlang jeder Kante. Da $h(s) = |V|$ und $h(t) = 0$, müsste ein erweiternder Pfad mindestens $|V|$ Kanten enthalten. Es gibt aber in einem Graphen mit V Knoten keine Pfade, die länger als $|V| - 1$ sind.

Korrektheits- und Komplexitätsbetrachtung I

- Wenn der Algorithmus terminiert, sind weder aktive Knoten noch erlaubte Kanten übrig.
- Keine aktiven Knoten impliziert, dass der Exzess an jedem Knoten verschwindet und der Pre-Fluss f daher ein Fluss ist.
- Im Restgraphen gibt es keinen Pfad von s nach t, also muss der Fluss (auf grund des min-cut-max-flow Theorems) maximal sein.

Korrektheits- und Komplexitätsbetrachtung I

Da in jedem **Push**-Schritt der Exzess reduziert wird, kann das nur endlich oft gemacht werden.

- Es sind $O(|E|)$ Kanten zu betrachten.

In jedem **Relabel**-Schritt wird ein $h(u)$ erhöht. Die länge von s-t-Pfade beträgt maximal $|V|-1$. Daher kann $h(u)$ nicht grösser als $2|V|$ werden, d.h., es können nicht mehr als $O(|V|^2)$ **Relabel**-Schritte auftreten.

$\Rightarrow$ Daher terminiert der push-relabel Algorithmus nach $O(|V|^2|E|)$ Schritten.

Einen vollständigen Beweis finden Sie hier: *(Link)*

Verbesserte Auswahl eines aktiven Knotens: Discharging

Bearbeite einen aktiven Knoten bis er inaktiv wird.

Algorithmus - Verbesserung

Solange Knoten aktiv ist:

- verwende **Push**-Schritte bis entweder der Knoten inaktiv ist oder alle erlaubten Kanten saturiert sind, i.e., die gesamte Restkapazität entlang der erlaubten Kanten aufgebraucht ist.
- mache einen **Relabel**-Schritt um neue erlaubte Kanten zu finden.

Ein solcher Schritt heißt **discharging**.

Verbesserte Auswahl eines aktiven Knotens: Discharging

Zusammenfassung

- Starte mit dem trivialen Pre-Fluss definiert durch die Kapazität der auslaufenden Kanten von s, $f(s,u) = c(s,u)$ und $f(x,y) = 0$ sonst. Konstruiere den Restgraphen G_f.
- Solange es einen Aktiven Knoten gibt: **discharge** ihn.

Die Performanz des Algorithmus verbessert sich weiter, wenn zunächst die aktiven Knoten mit dem größten Wert von h **discharged** werden.