

# Informatik II

## Übung 13

FS 2019

# Heutiges Programm

- 1 Feedback letzte Übung
- 2 Wiederholung Vorlesung
- 3 In-Class-Exercise (praktisch)

# 1. Feedback letzte Übung

## **2. Wiederholung Vorlesung**

# Fluss

Ein *Fluss*  $f : V \times V \rightarrow \mathbb{R}$  erfüllt folgende Bedingungen:

■ *Kapazitätsbeschränkung:*

Für alle  $u, v \in V$ :  $f(u, v) \leq c(u, v)$ .

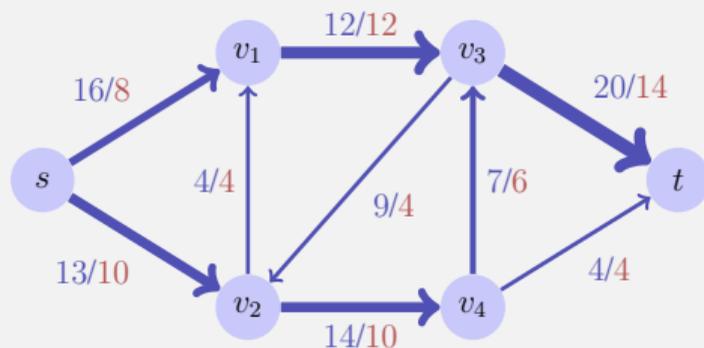
■ *Schiefsymmetrie:*

Für alle  $u, v \in V$ :  $f(u, v) = -f(v, u)$ .

■ *Flusserhaltung:*

Für alle  $u \in V \setminus \{s, t\}$ :

$$\sum_{v \in V} f(u, v) = 0.$$



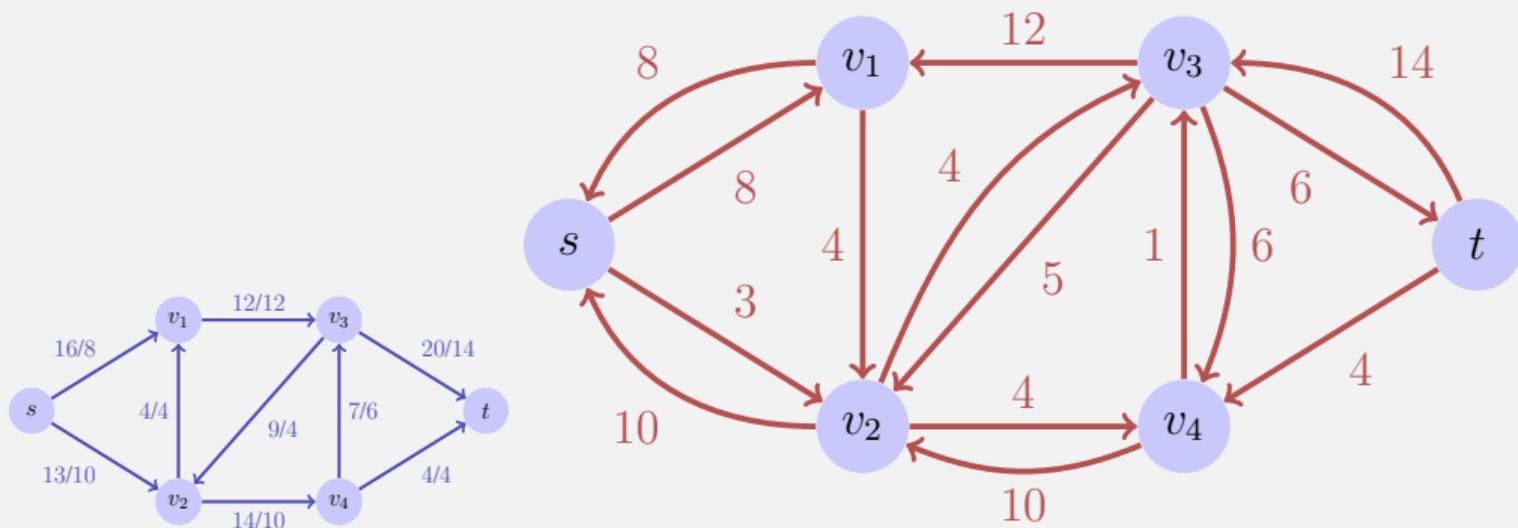
*Wert*  $w$  des Flusses:

$$|f| = \sum_{v \in V} f(s, v).$$

Hier  $|f| = 18$ .

# Restnetzwerk

*Restnetzwerk*  $G_f$  gegeben durch alle Kanten mit Restkapazität:



Restnetzwerke haben dieselben Eigenschaften wie Flussnetzwerke, ausser dass antiparallele Kapazitäten-Kanten zugelassen sind.

# Erweiterungspfade

*Erweiterungspfad*  $p$ : einfacher Pfad von  $s$  nach  $t$  im Restnetzwerk  $G_f$ .

*Restkapazität*  $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ Kante in } p\}$

# Max-Flow Min-Cut Theorem

## Theorem

*Wenn  $f$  ein Fluss in einem Flussnetzwerk  $G = (V, E, c)$  mit Quelle  $s$  und Senke  $t$  ist, dann sind folgende Aussagen äquivalent:*

- 1  $f$  ist ein maximaler Fluss in  $G$*
- 2 Das Restnetzwerk  $G_f$  enthält keine Erweiterungspfade*
- 3 Es gilt  $|f| = c(S, T)$  für einen Schnitt  $(S, T)$  von  $G$ .*

# Algorithmus Ford-Fulkerson( $G, s, t$ )

**Input:** Flussnetzwerk  $G = (V, E, c)$

**Output:** Maximaler Fluss  $f$ .

**for**  $(u, v) \in E$  **do**

└  $f(u, v) \leftarrow 0$

**while** Existiert Pfad  $p : s \rightsquigarrow t$  im Restnetzwerk  $G_f$  **do**

└  $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$

└ **foreach**  $(u, v) \in p$  **do**

└└  $f(u, v) \leftarrow f(u, v) + c_f(p)$

└└  $f(v, u) \leftarrow f(v, u) - c_f(p)$

# Praktische Anmerkung

In einer Implementation des Ford-Fulkerson Algorithmus werden die negativen Flusskanten normalerweise nicht gespeichert, da ihr Wert sich stets als der negierter Wert der Gegenkante ergibt.

$$f(u, v) \leftarrow f(u, v) + c_f(p)$$

$$f(v, u) \leftarrow f(v, u) - c_f(p)$$

wird dann zu

**if**  $(u, v) \in E$  **then**

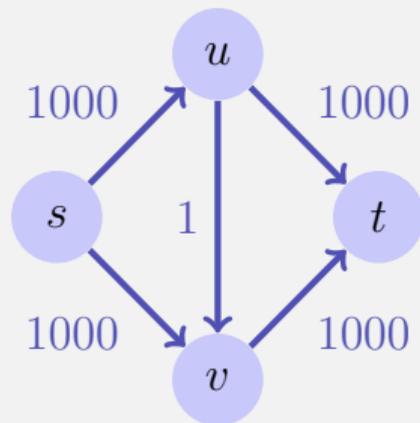
$$\quad | \quad f(u, v) \leftarrow f(u, v) + c_f(p)$$

**else**

$$\quad | \quad f(v, u) \leftarrow f(v, u) - c_f(p)$$

# Analyse

- Für ganzzahligen Fluss benötigt der Algorithmus maximal  $|f_{\max}|$  Durchläufe der While-Schleife (denn der Fluss erhöht sich mindestens um 1). Suche einzelner zunehmender Weg (z.B. Tiefensuche oder Breitensuche)  $\mathcal{O}(|E|)$ . Also Gesamtlaufzeit in  $\mathcal{O}(f_{\max}|E|)$ .



Bei schlecht gewählter Strategie benötigt der Algorithmus hier bis zu 2000 Iterationen.

# Edmonds-Karp Algorithmus

Wähle in der Ford-Fulkerson-Methode zum Finden eines Pfades in  $G_f$  jeweils einen Erweiterungspfad kürzester Länge (z.B. durch Breitensuche).

# Edmonds-Karp Algorithmus

## Theorem

*Wenn der Edmonds-Karp Algorithmus auf ein ganzzahliges Flussnetzwerk  $G = (V, E)$  mit Quelle  $s$  und Senke  $t$  angewendet wird, dann ist die Gesamtanzahl der durch den Algorithmus angewendete Flusserhöhungen in  $\mathcal{O}(|V| \cdot |E|)$ .*

*$\Rightarrow$  Gesamte asymptotische Laufzeit:  $\mathcal{O}(|V| \cdot |E|^2)$*

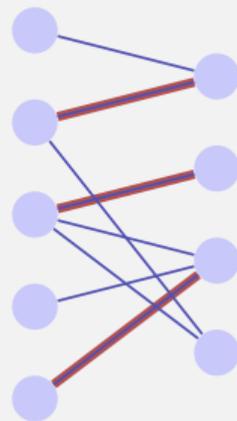
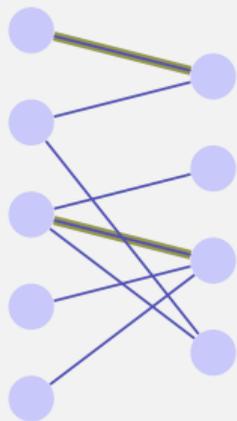
[Ohne Beweis]

# Anwendung: Maximales bipartites Matching

Gegeben: bipartiter ungerichteter Graph  $G = (V, E)$ .

**Matching**  $M$ :  $M \subseteq E$  so dass  $|\{m \in M : v \in m\}| \leq 1$  für alle  $v \in V$ .

**Maximales Matching**  $M$ : Matching  $M$ , so dass  $|M| \geq |M'|$  für jedes Matching  $M'$ .



# 3. In-Class-Exercise (praktisch)

Maximale Flüsse – Implementation

# Max-Flow Implementation

https://expert.ethz.ch/ide2/toCfBTjgRCdMFahMQ?fileKey=pkSZLA7Hbp7qwDrBH

ETH Code Expert Service Lectures at D... Datenstrukturen und ... D-BAUG Informatik II... dict.cc | Wörterbuch E... dict.leo.org - English... Linguee | Deutsch-Eng...

### Max Flow - Master Solution

```
31 Node endNode;
32
33 public Graph(int size){
34     this.size = size;
35     nodes = new Node[size];
36     capacities = new int[size][size];
37     residuals = new int[size][size];
38     flows = new int[size][size];
39     for (int i = 0; i < size; ++i)
40         nodes[i] = new Node(i);
41 }
```

Compilation successful  
number nodes:6  
edges [from to capacity], any neg

```
0 1 16
0 2 13
2 1 4
1 3 12
2 4 14
3 2 9
4 3 7
3 5 20
4 5 4
-1
```

flow = 0  
graph: 0 1 16|0 2 13|0 1 3 12|0  
flow = 12  
graph: 0 1 16|12 0 2 13|0 1 3 12|  
flow = 16  
graph: 0 1 16|12 0 2 13|4 1 3 12|  
flow = 23  
graph: 0 1 16|12 0 2 13|11 1 3 12|  
plot generated, please observe files

### Files

graph0.png

graph1.png

graph2.png

graph3.png

maximal flow of the given network.

The generated graphs are stored in a file and are visualized by a separate Python program.

Do thus not modify method `Graph.toFile`.

The following two example scenarios are contained as tests (when you hit the test button). Networks are defined by specification of

- the number of nodes followed by
- a sequence of edges: triples of source, destination and capacity
- terminated by a negative number

### Graph from the lecture:

```
6
0 1 16
0 2 13
2 1 4
1 3 12
2 4 14
3 2 9
4 3 7
3 5 20
4 5 4
-1
```

### Matching example from the lecture:

```
11
0 1 1
0 2 1
0 3 1
```

Fragen?