TU Berlin, Fakultät IV
Brock/Páll/Bhatt/Röhr/Stahl
MAR 4-3 und MAR 5-1

# Aufgabenblatt 9

*Flussgraphen und Minimale Schnitte*

## Abgabe (bis 04.07.2021 23:59 Uhr)

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im git Ordner eingecheckt sein: Die Hausaufgabenprüfung erfolgt automatisch am Donnerstag nach Mitternacht am 24.06.2021 und am 01.07.2021. Die folgenden Dateien werden für die Bepunktung berücksichtigt:

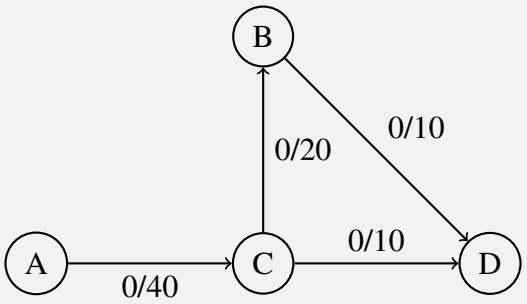
| **Geforderte Dateien:** | |
|---|---|
| `Blatt09/src/FlowApplications.java` | Aufgabe 3&4 |

Als Abgabe wird jeweils nur die letzte Version im git gewertet.

## Aufgabe 1: Flussgraphen (Tutorium)

Flussgraphen sind gerichtete Graphen mit Kantengewichten.

**Abbildung 1**




**1.1** Erläutern Sie die folgenden Begriffe anhand des Beispiel-Flussgraphen in Abbildung 1:

- Netzwerk, Flusskante, Kapazität
- Quelle, Senke
- Fluss, maximaler Fluss

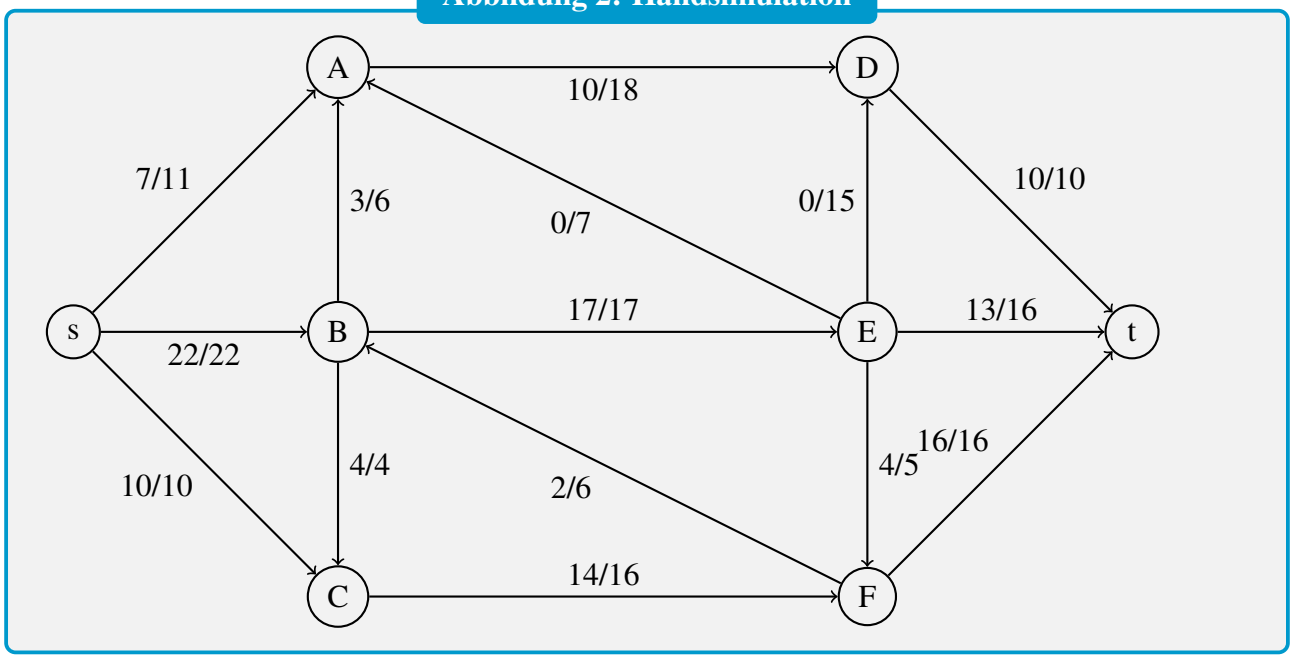
**1.2** Überlegen Sie sich Anwendungsbeispiele, in denen Flussgraphen relevant sein könnten.

**1.3** Was ist der Restflussgraph? Wie sieht der Restflussgraph für das Beispiel in Abbildung 1 aus?
**Alle weiteren Teilaufgaben beziehen sich auf die Abbildung 2.**

**1.4** Bestimmen Sie den Wert, den der Fluss in der Abbildung 2 hat.

Flusswert =

**Abbildung 2: Handsimulation**




**1.5** Führen Sie eine Iteration des Edmonds-Karp Algorithmus aus und notieren Sie die Knoten eines vergrößernden Pfades beginnend in $s$ und endend in $t$.

**1.6** Notieren Sie den Wert des maximalen Flusses des oben gezeigten Flussgraphen.

**1.7** Wo ist der minimale Schnitt? Schreiben Sie alle Knoten auf der Seite der Quelle des minimalen Schnittes auf.

**1.8** Notieren Sie die Kapazität des minimalen Schnittes in dem obigen Flussgraphen. Welcher Zusammenhang besteht zum maximalen Fluss?

# Aufgabe 2: Klassen für Flussgraphen studieren (Hausaufgabe)

Um die folgenden Aufgaben zu lösen, sollen Sie die vorgegebenen Klassen `FordFulkerson`, `FlowNetwork` und `FlowEdge` studieren. `FordFulkerson` generiert den maximalen Fluss in einem Flussgraphen mittels des Edmonds-Karp Algorithmus. Die anderen Vorgabe-Klassen stellen dafür die benötigten Datenstrukturen, Funktionen und Schnittstellen bereit.

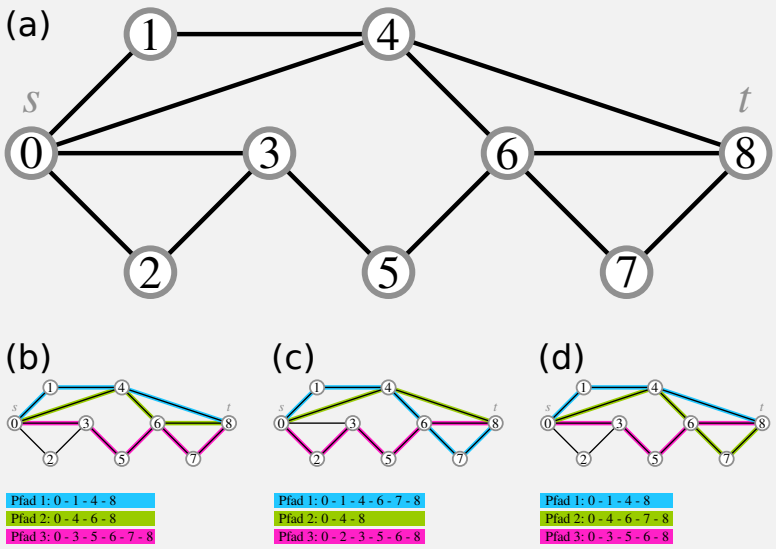
**Hinweis.** Beachten Sie, dass bei dem Aufruf des Konstruktors von `FordFulkerson` das als Eingabeargument übergebene `FlowNetwork` verändert wird (Seiteneffekt): In den Kanten ist der Fluss einer Konfiguration mit maximalem Fluss gespeichert.

# Aufgabe 3: Edmonds-Karp Anwendung (Hausaufgabe)

In dieser Aufgabe benutzen Sie den Edmonds-Karp Algorithmus, um ein allgemeines Graphen-Problem zu lösen, siehe Abbildung 3. Es sollen möglichst viele Kanten-disjunkte Pfade in einem gegebenen **ungerichteten** Graphen zwischen zwei Knoten $s$ und $t$ gefunden werden. Ein Menge von Pfaden heißt *Kanten-disjunkt*, wenn keine der Kanten von unterschiedlichen Pfaden benutzt wird. Knoten hingegen dürfen von unterschiedlichen Pfaden verwendet werden.

Im Folgenden geht es um Mengen Kanten-disjunkter Pfade mit einer maximalen Anzahl von Pfaden. Es kann unterschiedliche solcher maximaler Mengen geben, siehe Abbildung 3. In Aufgabe 3.1 soll die maximale Anzahl Kanten-disjunkter Pfade zwischen $s$ und $t$ herausgefunden werden und in Aufgabe 3.2 soll eine maximale Menge solcher Pfade zurückgegeben werden.

**Abbildung 3: Maximale Menge disjunkter Pfade**




Es sollen möglichst viele Pfade in einem ungerichteten Graphen zwischen zwei Knoten $s$ und $t$ gefunden werden, die keine gemeinsamen Kanten besitzen (Kanten-disjunkte Pfade). Für den Graphen (a) ist die maximale Anzahl Kanten-disjunkter Pfade 3. Es gibt unterschiedliche Möglichkeiten diese drei Pfade zu wählen, z. B. (b) bis (d).

## 3.1 Disjunkte Pfade zählen (20 Punkte)

Was ist die maximale Anzahl von Pfaden zwischen `s` und `t`, die untereinander Kanten-disjunkt sind. Implementieren Sie in der Klasse `FlowApplications` die Methode

```
int numberOfEdgeDisjointPaths(Graph graph, int s, int t)
```

die diese Anzahl zurückgibt. Überlegen Sie sich dazu, wie der gegebene Graph in einen speziellen Flussgraphen umgewandelt werden kann, in dem Frage nach der maximalen Anzahl Kanten-disjunkter Pfade durch den maximalen Fluss beantwortet wird.

**Hinweis.** Der Graph in Abbildung 3 (a) ist als Datei `Graph1.txt` gegeben. Sie können ihn benutzen, um Ihr Programm zu überprüfen. Beachten Sie, dass in den Tests auch Fälle mit `s≠0` und `t≠V-1` geprüft werden.

## 3.2 Disjunkte Pfade zurückgeben (20 Punkte)

In dieser Aufgabe soll eine maximale Menge disjunkter Pfade zwischen den Knoten `s` und `t` in einem ungerichteten Graphen erstellt werden. Dazu soll die Methode

```
Bag<LinkedList<Integer>> edgeDisjointPaths(Graph graph, int s, int t)
```

in der Klasse `FlowApplications` implementiert werden. Offensichtlich kann mit dieser Aufgabe die vorige Aufgabe direkt gelöst werden (die Anzahl ist `paths.sizeof()`, wenn `paths` die Rückgabe von `edgeDisjointPaths()` ist). Allerdings ist diese Aufgabe deutlich schwieriger. Daher ist es empfehlenswert, zunächst die vorige Aufgabe separat zu lösen.

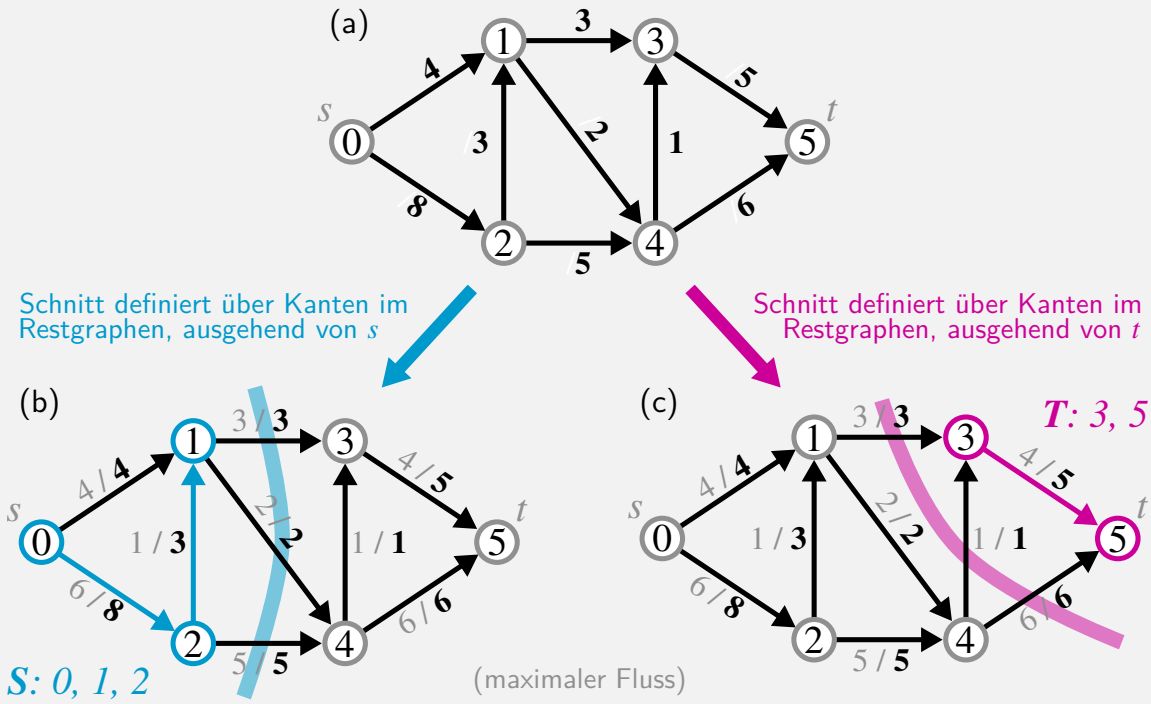
Die gesuchten Pfade können aus dem Flussgraphen, der zur Lösung der vorherigen Aufgabe erstellt wurde, abgelesen werden.

# Aufgabe 4: Eindeutigkeit des minimalen Schnittes (Hausaufgabe)

## 4.1 Eindeutigkeit minimaler Schnitte prüfen (40 Punkte)

In der Vorlesung wurden minimale Schnitte (*mincuts*) definiert und gezeigt, dass die Kapazität des minimalen Schnittes mit dem maximalen Fluss übereinstimmt. Für einen Flussgraphen können allerdings mehrere Schnitte existieren, die die minimale Kapazität haben, siehe Abbildung 4. Der minimale Schnitt ist also nicht immer eindeutig.

**Abbildung 4: Minimale Schnitte und Eindeutigkeit**




Die Kapazität eines $s$-$t$ Schnittes ist die Summe der Gewichte (bzw. Kapazitäten) aller kreuzender Kanten in Richtung von $s$ nach $t$ (genaue Definition siehe Vorlesung). Minimale Schnitte können in einem Flussgraphen mit maximalem Fluss gefunden werden. Es kann auch mehrere minimale Schnitt geben. Definiert man die Menge $\boldsymbol{S}$ als alle von Knoten $s$ im Restgraphen erreichbare Knoten, so stellen $\boldsymbol{S}$ und sein Komplement einen minimalen Schnitt dar, siehe (b). Auch die Menge $\boldsymbol{T}$ aller Knoten, von denen $t$ im Restgraphen erreicht werden kann, definiert einen minimalen Schnitt, siehe (c). In dem gezeigten Beispiel gibt es unterschiedliche minimale Schnitte. Falls in einem Graphen, die durch $\boldsymbol{S}$ und $\boldsymbol{T}$ definierten Schnitte übereinstimmen, so ist der minimale Schnitt eindeutig.

In dieser Teilaufgabe soll eine Methode geschrieben werden, die prüft, ob zu einem gegebenen Flussgraphen mehrere minimale Schnitte exitieren, oder ob er eindeutig ist. Dazu können Sie den folgenden Test verwenden (dessen Korrektheit hier nicht bewiesen wird). Es wird ein zweiter minimaler Schnitt basierend auf dem Restgraphen eines maximalen Fluss definiert. Der eine Schnitt wird ausgehend von $s$ in Kantenrichtung, der andere ausgehend von $t$ gegen Kantenrichtung berechnet. Wenn diese beiden Verfahren zu demselben Schnitt führen, ist er eindeutig, siehe Abbildung 4 und den Beweis von *Vergrößernde Pfade und maximaler Fluss* aus der Vorlesung.

Implementieren Sie dann die Methode

```
boolean isUnique(FlowNetwork flowNetwork, int s, int t)
```

**Hinweise:**

- Verwenden Sie zwei boolsche Arrays, um die beiden Schnitte darzustellen und prüfen Sie am Ende, ob die beiden Arrays denselben Schnitt repräsentieren.
- Der in Abbildung 4 abgebildete Beispielgraph ist als Datei `Flussgraph1.txt` gegeben. Die Datei `Flussgraph2.txt` repräsentiert einen Flussgraphen mit eindeutigem minimalen Schnitt (ebenfalls für `s=0` und `t=5`. Beachten Sie, dass in den Tests auch Fälle mit `s≠0` und `t≠V-1` geprüft werden.

## 4.2 Engpässe (20 Punkte)

In einem Flussgraphen nennt man eine **Kante** einen Engpass (*Bottleneck*), falls die Erhöhung ihrer Kapazität zu einer Vergrößerung des maximalen Flusses führt. Die **Knoten**, die Startknoten einer Engpass-Kante sind, nennt man Engpass-Knoten. Wenn man (gegen die Regeln!) bei schon maximalem Fluss, den Fluss weiter erhöhen würde, käme es an diesen Knoten zum 'Stau' oder Überlauf. Daher ist das Erkennen der Engpass-Knoten eine wichtige Funktion.

Wenn ein Flussgraph einen eindeutigen minimalen Schnitt besitzt (siehe erster Aufgabenteil), dann können die Engpass-Kanten (und damit auch die Engpass-Knoten) leicht identifiziert werden. Es sind die Kanten, die den minimalen Schnitt kreuzen. (Im allgemeinen Fall sind es die Kanten mit Startknoten in $S$ und Endknoten in $T$ mit den Bezeichnungen aus Abbildung 4.)

Implementieren Sie die Methode

```
LinkedList<Integer> findBottlenecks(FlowNetwork flowNetwork, int s, int t)
```

die eine Liste der Indizes aller Engpass-Knoten in `flowNetwork` zurückgibt. Es darf vorausgesetzt werden, dass der übergebene Flussgraph einen eindeutigen minimalen Schnitt besitzt. (Falls dies nicht zutrifft, sollte eine Exception geworfen werden. Um die Aufgabenteile unabhängig voneinander zu bewerten, wird dies nicht in den Tests überprüft.)