Inhaltsverzeichnis

[GKA-Praktikum3 1](#_Toc469299692)

[Algorithmen 1](#_Toc469299693)

[FordFulkerson 1](#_Toc469299694)

[getNextNode() 2](#_Toc469299695)

[inspiziere(Knoten) 2](#_Toc469299696)

[vergroessereWeg() 3](#_Toc469299697)

[markiereKnotenAnKante(knoten, kante) 3](#_Toc469299698)

[EdmondsKarp 4](#_Toc469299699)

[getKuerzesterWeg() 4](#_Toc469299700)

[getBTSVorgaenger(Node) 4](#_Toc469299701)

[Fragen aus der Aufgabe 5](#_Toc469299702)

[Testkonzept 5](#_Toc469299703)

# GKA-Praktikum3

Es sollen die Algorithmen FordFulkerson und EdmondsKarp implementiert werden. Anschließend sollen sie gegeneinander antreten. Das antreten wird in diesem Fall ausgelassen, da EdmondsKarp noch nicht funktioniert.

Das Projekt wurde wie folgt aufgeteilt:

Entwicklung Generator: Seemann

Implementierung der Algorithmen: Höling

JUnit: Seemann

Dokumentation: Höling/Seemann

# Algorithmen

## FordFulkerson

Der FordFulkerson inspiziert jeden Knoten bei der Quelle beginnend und berechnet das minimale Delta auf einem Weg zur Senke. Wenn ein erweiternder weg gefunden wurde (das Delta auf dem weg ist > 0), dann wird das Delta auf alle Knoten des Weges addiert. Das inspizieren wird so lange wiederholt bis keine erweiternder Weg mehr gefunden wurde.

FordFulkerson(Graph)  
Input: Der graph auf dem Operiert werden soll

Output: liefert den maximalen Fluss  
maximalerFluss := ein Zaehler der alle Deltas aufaddiert  
  
initialisierung();

do{  
 KnotenMomentan := getNextNode();  
 IF KnotenMomentan != null  
 inspiziere(KnotenMomentan);  
 ELSE  
 //es gibt keinen vergrößernden Weg  
 break;  
 END IF

IF die Senke ist markiert  
 vergroessereWeg()  
 END IF  
}while es sind nicht alle Knoten inspiziert

## getNextNode()

Liefert einen Knoten zurück, der markiert aber nicht inspiziert ist.

getNextNode()  
Output: Ein Knoten, der markiert, aber NICHT ispiziert istFOR alle Knoten aus dem Graphen  
 IF der knoten ist markiert und nicht inspiziert  
 return knoten;  
 END IF  
END FOR  
  
return null;

## inspiziere(Knoten)

Der übergebene Knoten wird inspiziert, indem über alle Kanten des Knotens iteriert wird und der gegenüberliegende Knoten wird dann markiert.

Input: ein Knoten der markiert aber nicht inspiziert ist  
  
FOR jede Kante am Knoten  
 markiereKnotenAnKante(knoten, Kante);  
END FOR  
  
der Knoten wird als inspiziert markiert;

## vergroessereWeg()

Wenn es einen vergroessernden Weg gibt, werden die Kanten jeweils um das „mitgeschleifte“ delta erhöht.

momentanKnoten := die Senke;  
deltaSenke := ist das Delta was momentan in der Senke steht  
  
WHILE der momentaneKnoten ist nicht die Quelle  
 vorgaenger := schaue im aktuellen Knoten und lese den Vorgänger aus  
 Kante := die Kante zwischen dem momentanKnoten und dem Vorgänger  
 addiereKantenFluss mit deltaSenke  
 momentanKnoten := vorgaenger  
END WHILE  
  
FOR jeder Knoten im Graphen  
 entferne die Markierung  
 entferne markierung inspiziert  
END FOR  
  
markiere die quelle;

## markiereKnotenAnKante(knoten, kante)

Markiert einen anliegenden knoten mit dem neuen Delta  
  
Input : knoten – der Knoten „von dem man aus kommt“  
 kante – die Kante die auf dem weg von der Senke zur Quelle liegt  
Output: keiner  
  
gegenüber := der gegenüberliegende Knoten auf der Kante  
  
IF gegenüber ist NICHT markiert  
 IF kante ist eine Rückwärtskante  
 kantenfluss := getF(kante);  
 kantenKapazität := getC(kante);

IF kantenfluss < kantenkapazität  
 knotenFlussVi := getF(knoten);  
 markiere gegenüber mit dem Vorgänger „knoten“  
 markiere gegenüber mit dem fluss min(kantenkapazität-kantenFluss),knotenflussVI)

END IF  
 ELSE  
 IF kantenfluss > 0.0  
 markiere gegenüber mit dem Vorgänger „knoten“  
 markiere gegenüber mit dem fluss min(kantenkapazität-kantenFluss),knotenflussVI)  
 END IF  
END IF

# EdmondsKarp

Der EdmondsKarp arbeitet wie der FordFulkerson. Er vergrößert den vergrößernden weg anhand des kürzesten wegen (BTS).

InitialisiereBTS();  
breitensuche();

kuerzesterWeg := eine Liste die den kürzesten Weg enthält beginnend bei q  
  
DO  
 kuerzesterWeg = getKuerzesterWeg();  
 FOR jede Kante in küerzesterWeg  
 markiereKnotenAnKante(kante, kante.sourceNode());  
 END FOR  
WHILE die länge des Weges des kürzestenWeg ist größer 0;

## getKuerzesterWeg()

Aufgrund der Breitensuche wird der kürzeste Weg rekonstruiert. Wenn die Kante die zum kürzesten Weg hinzugefügt werden soll, wird geprüft ob sie voll ist, also fluss=kapazitaet. Dann wird diese Kante übersprungen und das nächste minimum gesucht.

momentanKnoten := der Knoten der Momentan untersucht wird  
momentanKnoten := Senke;

WegListe := eine Liste die den weg enthalten

WHILE der momentanKnoten ist nicht q  
 vorgaenger := getBTSVorgaenger(momentanKnoten)  
 kante := kante zwischen momentanKnoten und vorgaenger  
 wegListe.add (kante);  
 momentanKnoten := vorgaenger;  
END WHILE  
  
return wegListe;

## getBTSVorgaenger(Node)

liefert den Vorgänger eines Knotens, wenn die dazwischen liegende Kante nicht voll ist.

Input: der Knoten dessen vorgaenger geliefert werden soll  
Output: die Kante mit dem Vorgängerknoten

Minimumgewicht := das minimale Gewicht der anliegenden Knoten  
minimumKnoten := der Knoten mit dem minimalen Gewicht  
minimumKante := die kante am minimalen Knoten

FOR jede Kante am übergebenen Knoten  
 vorgaenger := kante.getSource()  
 vorgaengerGewicht := vorgaenger.gewicht

IF kante ist nicht voll  
 IF vorgaengerGewicht < minimumgewicht  
 minimumKnoten := vorgaenger  
 minimumGewicht := minimumKnoten.gewicht  
 minimumKante := kante  
 END IF  
 END IF  
END FOR  
  
return minimumKante;

# Fragen aus der Aufgabe

**Welcher Algorithmus/welche Implementierung ist schneller? Wie schnell für die Netzwerke Ihrer Praktikumsgruppe?**

**Was haben Sie unternommen, um eine bessere Laufzeit zu erreichen?**

**Lässt sich die Laufzeit Ihrer Implementierung durch andere Datenstrukturen verbessern?**

Definitiv. Es könnte beim wählen des zu inspizierenden Knotens ein HashSet verwendet werden, anstatt die Informationen ob ein Knoten markiert oder inspiziert ist direkt als Attribut in dem Knoten zu speichern.

**Was passiert, wenn Sie nicht-ganzzahlige Kantengewichte wählen?**

Beim Auslesen eines Graphen aus einer Datei werden nur ganzzahlige Werte ausgelesen. Wenn dies möglich wäre, könnte der Algorithmus aber mit nicht-ganzzahligen Gewichten umgehen.

**Was passiert bei negativen Kantengewichten?**

Das funktioniert nicht, da bei den Flüssen geprüft wird ob ein Fluss > 0 ist. Bei der Prüfung ob der Fluss < Kapazität ist, würde die Prüfung immer fehlschlagen.

# Testkonzept

Der Ford-Fulkerson ist bei uns nur an zwei Stellen zu testen. Die eine ist der Maximale Fluss, der am Ende herauskommt und die andere Stelle ist die Laufzeit. Diese können wir aber nur händisch testen. Der Edmond-Karp funktioniert fast genauso wie der Ford-Fulkerson und ist daher genauso zu testen. Um die dauerhafte Lauffähigkeit zu garantieren lassen wir die Algorithmen auf einen zufällig erstellten Graphen laufen und zwar wie gefordert 100 Mal. Der zufällig erstellte Graph kann in beliebiger Größe erstellt werden. Um die korrekte Erstellung zu garantieren, wird der Graph ebenfalls auf getestet.

Allgemein kann man noch sagen, dass wir um eine höhere Testabdeckung zu erhalten, testen wir unsere Algorithmen auf mehreren unterschiedlichen Graphen.