**Team**: <3_4>, <Patrick Steinhauer, Jan Dennis Bartels>

**Aufgabenaufteilung**:

Hauptsächlich Patrick Steinhauer aufgrund von Krankheit.

Kleinere Sachen hat mein Partner aber noch gemacht gehabt z.B. die Erstellung des Netzwerkes.

**Quellenangaben**:

Da das Math.Random nicht so funktionierte wie wir wollten haben wir uns eine Random Funktion aus dem Internet gesucht.

http://blog.root-of-all-evil.com/2010/03/math-random-zufallszahlen-in-java/

Klasse markedVertex von der Gruppe Tim Hagemann und Sebastian Diedrich

**Bearbeitungszeitraum**: Bearbeitungszeit leigt bei ca 20-30 Stunden

**Aktueller Stand**:

Die Aufgabe ist fertig implementiert.

Da wir aufgrund von Krankheit meines Partners gesagt hatten dass nur der Ford Fulkerson fertig sein sollte funktioniert dieser nun.

Edmond karp werde ich aus eigenem interesse wahrscheinlich noch zusätzlich machen.

Wir haben zwei Versionen der Aufgabe, eine mit einer erstellten Klasse einer nachbargruppe.

Und eine die selbst überlegt wurde.

Die Klasse wurde von der Gruppe mit Sebastian Diedrich übernommen.

**<u>Beantwortung der Fragen:</u>**

1. Können wir nicht genau beantworten, da uns der zweite Algorithmus fehlt. Jedoch habe ich im Internet gelesen, dass der Edmond Karp schneller sein soll.

2. Um unsere Laufzeit zu verbessern haben wir aus unserer ersten Version vom Praktikum einige Sachen entfernt, sowie andere Datenstrukturen verwendet um besser arbeiten zu können.

3. Dies geht , indem für spezielle Operationen die richtige Datenstruktur verwendet wird. z.B für sortierte Sachen eine Datenstruktur verwenden, die sofort sortiert von sich aus.

4. Bei nicht ganzzahligen Kantengewichten kommt es sehr wahrscheinlich zu einem falschen Ergebnis, da so möglicherweise gar kein passender größter Weg gefunden werden kann. Meiner Meinung nach gibt es deswegen entweder ein falsches Ergebnis oder eine nicht Terminierung des Algorithmus.

5. Dies sollte im Prinzip bei dem Algorithmus nicht möglich sein, da hier vorgesehen ist, dass dieser auf den nicht negativen reelen Zahlen arbeitet. Wenn also eine negative Zahl an einer Kante stehen würde, würde der Algorithmus wohl ebenfalls nicht zum stehen kommen.

**generateBigNet** ->

Die Größe der erstellten Netzwerkes kann mit den Klassenvariablen angepasst werden.
Standartmäßig werden 40 Vertexe und 40 Kanten erstellt

Die wirkliche Größe des Netzwerkes variiert im +- 15%

Die Vertexe werden einfach hinzugefügt.

Die Kanten werden in mehreren Schritten hinzugefügt.

Zuerst wird die quelle über eine zufällige Anzahl von Vertexen mit der Senke verbunden.

Danach werden zufällige Vertexe miteinander verbunden bis eine berechnete Kantenanzahl erreicht ist.

Die Gewichtung der Kanten ist zufällig.

Tests:

Getestet haben wir auf die Funktionsweise und Korrektheit des Algorithmus.

Hierfür haben wir verschiedene Test Durchläufe gehabt die dies testen.

Weiterhin haben wir uns zum testen z.B. an den Graphen aus den Vorlesungsfolien gehalten haben.

In den Folien war dieser unter Aufgabe 3 zu finden Vorlesung 7.

Dieser wurde hauptsächlich benutzt um komplett nachvollziehen zu können, ob der Algoritmus voll funktionsfähig ist.

Ford Fulkerson:

Unser Ford Fulkerson Algorithmus arbeitet nicht mit einer Fluss Angabe an dem Graphen direkt.

Die Flüsse werden jeweils nur berechnet, indem die Kapazität gegeben ist. Hierbei wird der Fluss Wert immer in einer Zweifachgeschachtelten Map gespeichert, die den source Vertex beinhaltet,

sowie den Target Vertex und den Flusswert. Des Weiteren ist der Algorithmus diesmal an den Vorlesungsfolien implementiert worden, was zum Verständnis im nach hinein sehr gut war.

Bei der Implementation habe ich versucht mit verschiedenen Datenstrukturen zu arbeiten, wobei zum Schluss doch hauptsächlich Listen, und Maps verwendet wurden.

Maps aufgrund der guten Schachtelbarkeit, und Listen, um Elemente schnell und leicht hinzuzufügen zu können und wieder entfernen zu können.

In einer der Zwei Implementierungen habe ich die Klasse MarkedVertex der Nachbargruppe verwendet, da mir diese Methode persönlich nicht bekannt war um so Werte speichern zu können.

Dies hat sehr für den Lerneffekt zum programmieren beigetragen und war ebenfalls hilfreich für so manches Problem.

In der zweiten Variante habe ich das Problem mit einer Speziell geschachtelten Map umgangen, die sowohl das Inkrement beinhaltet, sowie die Richtung der Edge. (+v3, 3)