GKAP Praktikum 04 15.06.2015

**– Team:**

Teamnummer 2\_5 - Dimitri Meier, Saeed Shanidar

**– Aufgabenaufteilung**:

1. Aufgaben, für die Dimitri Meier verantwortlich ist:
   1. Fleury Algorithmus
   2. Hierholzer Algorithmus
   3. Eulergraph-Generator
   4. Tests
2. Aufgaben, für die Saeed Shanidar verantwortlich ist:
   1. Dokumentation
   2. Theorie Aufgaben
   3. Eulergraph-Generator
3. Aufgaben, für die Dimitri Meier und Saeed Shanidar verantwortlich sind:
   1. Fleury Algorithmus
   2. Hierholzer Algorithmus
   3. Eulergraph-Generator
   4. Tests

**– Quellenangaben:**

Breitensuche (BFS)-Algorithmus <http://de.wikipedia.org/wiki/Breitensuche>

Dijkstra-Algorithmus: <http://de.wikipedia.org/wiki/Dijkstra-Algorithmus>

A\*-Algorithmus: [http://de.wikipedia.org/wiki/A\*-Algorithmus](http://de.wikipedia.org/wiki/A*-Algorithmus)

Prim-Algorithmus: <http://de.wikipedia.org/wiki/Algorithmus_von_Prim>

Kruskal-Algorithmus: <http://de.wikipedia.org/wiki/Algorithmus_von_Kruskal>

http://www-m9.ma.tum.de/Allgemeines

Fleury-Algorithmus: <http://de.wikipedia.org/wiki/Eulerkreisproblem#Algorithmus_von_Fleury>

Hierholzer-Algorithmus: <http://de.wikipedia.org/wiki/Algorithmus_von_Hierholzer>

Zusätzlich Die Vorlesungsfolien:

* <https://pub.informatik.haw-hamburg.de/home/pub/prof/padberg_julia/Home_GKA_SoSe15/Folien/vl04.pdf>
* <https://pub.informatik.haw-hamburg.de/home/pub/prof/padberg_julia/Home_GKA_SoSe15/Folien/vl05.pdf>

**- Verwendete Library:**

Java Universal Network/Graph Framework: JUNG <http://jung.sourceforge.net/>

Hipster heuristic search for Java: <http://www.hipster4j.org/>

JGraphT: <http://jgrapht.org/>

**– Aktueller Stand:**

* **GUI**: wurde erweitert
* **Algorithmus**:
  + BFS fertig Implementiert.
  + Dijkstra fertig Implementiert
  + A\* fertig Implementiert
  + Kruskal fertig Implementiert
  + Prim fertig Implementiert
  + Fibonacci-Heap fertig Implementiert
  + Fleury fertig Implementiert
  + Hierholzer fertig Implementiert
* **Graph-Generator**
  + Ungerichtet & Gewichtet-Generator fertig Implementiert
  + Ungerichtet & Gewichtet & Attribuiert-Generator fertig Implementiert
  + Zusammenhängende Ungerichtet & Gewichtet-Generator fertig Implementiert
  + Eulergraph-Generator fertig Implementiert
* **Tests:** 
  + GraphBuilder und den Algorithmus ausführlich getestet.
  + Test auf randomisierte unterschiedlich große Graphen mit Kruskal und Prim, wobei die Gesamtweglänge der Ergebnis Spannbaume gleich sein müssen.
* **Dokumentation:** 
  + Praktikums 01 : Fertig
  + Praktikums 02 : Fertig
  + Praktikums 03 : Fertig
  + Praktikums 04 : Fertig

### Eulerscher Graph Generator

### Die Klasse UndirectedEulerianGraphGenerator hat die Klassenmethode generatorGraph(), die uns einen zusammenhängenden, ungerichteten Eulerschen Graph generieren soll.

### Idee der Grapherzeugung Als erstes erzeugen wir eine Menge von Knoten. Anschließend unterteilen wir die Knoten in Untermenge, wobei eine Untermenge nicht kleiner 3 und nicht größer “Knotenmenge/4“ seien darf. Im Falle einer ungeraden Bruchrechnung auf Knotenmengen, werden die restlichen Knoten zu einen der bereits erstellten Untermengen angehängt. Die daraus entstehenden Untermengen werden nun je Untermenge kreisförmig mit einander verbunden, so dass alle Untermengen disjunkte Kreise ergeben.

### Die Vorteile von Kreisen sind, dass diese zusammenhängend und einen geraden Knotengrad besitzen. Diese Eigenschaft benötigen wir auch für unseren Eulerschen Graph. Nun sollen die disjunkten Kreise mindestens eine Verbindung zu einen anderen disjunkten Kreis besitzen, so dass wenigstens jeder Kreis mit irgendeinem anderen verbunden ist. Um ein wenig Zufälligkeit im Ergebnisgraphen zu haben, haben wir eine Maximalgrenze zu den Verbindungen unter den Kreisen eingefügt. In unserm Fall Knotenanzahl \* 10. Zum Schluss müssen wir nun ein wenig korrigieren, da wir eventuell ungeraden Knotengrad im Graphen besitzen. Wir holen uns eine Liste von Knoten, die einen ungeraden Knotengrad besitzen. Jetzt wird ein zufälliger Knoten aus dieser Liste entnommen und mit einen anderen dieser Liste verbunden. Dies wird solange ausgeübt, bis diese Liste leer ist. Jetzt ist unser Graph fertig und besitzt die notwenigen Eigenschaften.

**Algorithmus von Fleury**

Im Algorithmus von Fleury spielen Brückenkanten eine wichtige Rolle. Das sind Kanten, ohne die der Graph in zwei Zusammenhangskomponenten zerfallen würde.

* Vorbedingung: Der übergebene Graph muss ein Eulerscher Graph sein.
* Nachbedingung: Der zurückgegebene Kreis ist tatsächlich ein Eulerscher Kreis

Der Algorithmus fügt einer anfangs leeren Kantenfolge alle Kanten eines Graphen hinzu, sodass ein Eulerkreis entsteht.

1. Wähle einen beliebigen Knoten als aktuellen Knoten.
2. Wähle unter den unmarkierten, mit dem aktuellen Knoten inzidenten Kanten eine beliebige Kante aus. Dabei sind zuerst Kanten zu wählen, die im unmarkierten Graphen keine Brückenkanten sind.
3. Markiere die gewählte Kante und füge sie der Kantenfolge hinzu.
4. Wähle den anderen Knoten der gewählten Kante als neuen aktuellen Knoten.
5. Wenn noch unmarkierte Kanten existieren, dann gehe zu Schritt 2.

Ob eine Kante eine Brückenkante ist, kann mittels Bereitensuche(BFS)-Algorithmus in Laufzeit O(|E|) überprüft werden.

Da pro Schritt eine Kante markiert und Imaginär entfernt wird, benötigen wir |E| Iterationen. Die Anzahl der pro Iteration geprüften Kanten entspricht dem Grad des aktuellen Knotens. Insgesamt kann man die gesamte Anzahl überprüfter Kanten durch O(|E|) beschränken. Die gesamte Laufzeit ist damit von der Größenordnung O(|E|)².

**Algorithmus von Hierholzer**

Mit dem Algorithmus von Hierholzer lässt sich eine Euler-Tour in einem eulerschen Graphen G(V; E) bestimmen. Der Algorithmus benutzt die Tatsache, dass sich eulersche Graphen in schrittweise Kantendisjunkte Zyklen zerlegen lassen.

* Vorbedingung: Der übergebene Graph muss ein Eulerscher Graph sein.
* Nachbedingung: Der zurückgegebene Kreis ist tatsächlich ein Eulerscher Kreis

Voraussetzung:

Es wird ein Eulerscher Graph erwartet mit einen geraden Knotengrad.

1. Der Eingabegraph muss zusammenhängend sein.
2. Der Eingabegraph muss ungerichtet sein.
3. Der Konten grad muss gerade sein.

Implementierung:

**Tests**

1. Für den Fleury und Hierholzer haben wir zu Anfang einen Händischen Eulerschen Graphen erzeugt und diesen auf Papier ausgewertet. Als Information für den Test des Algorithmus haben wir uns die Anzahl der Kanten vom Eulerkreis notiert.

Im Testfall wurde dann dieser Graph aus der Datei geladen und ausgeführt. Das Ergebnis sollte dann der erwartete Wert sein.

Dieser Test wurde dann in einer Schleife realisiert, so dass wir alle Knoten als Start-Knoten einmal benutzt haben. Das Ergebnis muss dann ebenfalls stimmen.

1. Dann haben wir uns Gedanken gemacht wie man noch besser testen könnte.  
   Wir sind auf die Idee gekommen, die Kanten des Eulerschen Kreises zu testen.  
     
   Somit packen wir die Kantenliste des Eulershen Kreises in ein Set um eventuell doppelt besuchte Kanten festzustellen. Dieses Set vergleichen wir nun an der Anzahl der Kanten im tatsächlichen Graphen.

Wenn die Anzahl gleich ist, dann hat der Algorithmus keine Kanten doppelt besucht.

1. Wichtig ist noch zu prüfen, ob diese Kantenfolge des Eulerschen Kreises auch zusammenhängend ist. Somit müssen wir diese Prozedur mit Hilfe von unseren ADT-Graphen lösen.   
     
   Dieser Test wird nun auf 100 verschiedene Randomisierte Eulersche Graphen ausgeführt. Sobald einer der oben beschriebenen Tests fehlschlägt, wird der Graph in einen gesonderten Bereich auf der Festplatte gespeichert.   
     
   Dadurch hat man die Möglichkeit später den eventuellen Fehlerfall wiederherzustellen und zu beheben.

**Theorieteil 4**

Aufgabe X:

1. Geben Sie bitte mit Begründung einen zusammenhängenden Graphen an, der einen Hamiltonkreis, aber keinen Eulerkreis enthält.

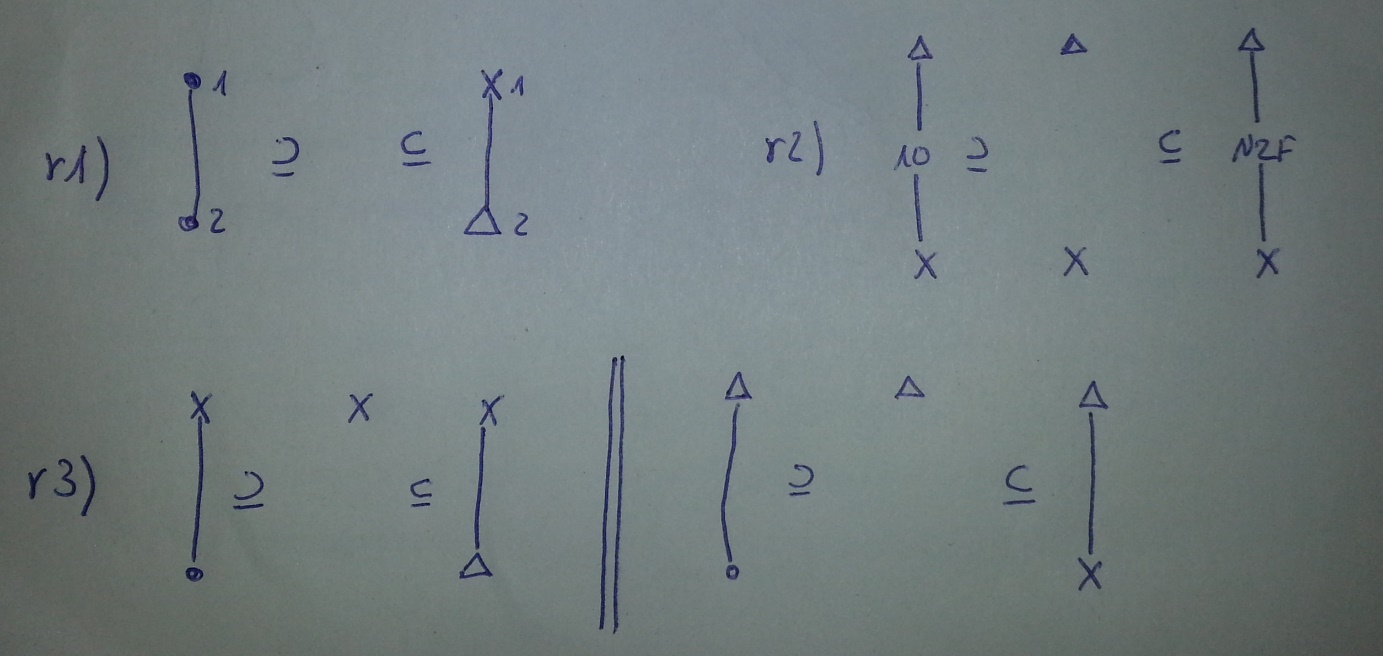
Bei **Hamiltonkreis** ist ein geschlossener Pfad in einem Graphen, der jeden Knoten genau einmal enthält. Im Gegensatz zum **Eulerkreis**, bei dem alle Kanten genau einmal durchlaufen werden

1. Geben Sie bitte mit Begründung einen zusammenhängenden Graphen an, der einen Eulerkreis, aber keinen Hamiltonkreis enthält.

Bei **Hamiltonkreis** ist ein geschlossener Pfad in einem Graphen, der jeden Knoten genau einmal enthält. Im Gegensatz zum **Eulerkreis**, bei dem alle Kanten genau einmal durchlaufen werden

Aufgabe XI:

1. Geben Sie eine Graphgrammatik 2F an, die für einen gegebenen Startgraphen G eine 2-Färbung erzeugt, wobei die Regeln solange wie möglich angewendet werden. Wenn es keine 2-Färbung gibt, dann soll ein Knoten mit dem Label N 2 F erzeugt und abgebrochen werden.



Aufgabe XII:

Die Brauerei braut Bier und stellt die Fässer in ihr kleines Lager. Das Lager der Brauerei fasst jedoch nur 40 Fässer. Die Fässer werden mit einem der drei Pferdewagen zum Gasthof trans-

portiert. Ein Pferdewagen transportiert genau 10 Fässer Bier. In der Gaststätte lassen sich aus einem Fass 50 Gläser Bier zapfen. Die Kellnerin kann maximal 6 Gläser tragen, geht aber nur los, wenn mindestens 3 Gläser gefüllt auf dem Tresen stehen.

Modellieren Sie dieses Szenario bitte mit Hilfe eines Stellen/ Transitionsnetzes