GKA Lösungsdokumentation Aufgabe 3

von Leonhard Pahlke, Jannik Sturhann (GKAP04_TeamC) und Chau Vu

Aufgabenaufteilung:
– Jannik Sturhann (Hierholzer + Tests)
– Leonhard Pahlke (Graph Generator + Tests
– Chau Vu (Fleury + Tests)

Bearbeitungszeitraum:
- 14.06.2019 – 16.06.2019 Hierholzer und Tests
- 12.05.2019 – 16.06.2019 Graph Generator und Tests
- 14.06.2019 – 16.06.2019 Fleury und Tests

Quellenangaben:
– Graphstream core - Stellt Graph Datenstruktur bereitstellt, die visuell dargestellt werden kann und auf der gearbeitet werden kann. (github.com/graphstream/gs-core)
– GraphStream JavaFX - Bietet Schnittstelle zwischen GraphStream und JavaFx Oberflächen (github.com/graphstream/gs-ui-javafx)
– Fleury Algorithmus: https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/

Inhaltsverzeichnis

Contents

Hierholzer Algorithmus 1
Implementierung 2
Tests 3
Fleury Algorithmus 3
Eulergraph Generator 5

Hierholzer Algorithmus

Algorithmus um in einem zusammenhängenden Graphen, in dem alle Knoten einen geraden Knotengrad haben, ein Eulerkreis zu finden. Ein Eulerkreis ist ein Kreis in einem Graphen, der jede Kante des Graphen genau einmal enthält

Algorithmus

Voraussetzung:
Sei $G = (V, E)$ ein zusammenhängender Graph, der nur Knoten mit geradem Grad aufweist.

i. Wähle einen beliebigen Knoten v_0 des Graphen und konstruiere von v_0 ausgehend einen Unterkreis K in G, der alle Eigenschaften eines Eulerkreises besitzt.

ii. Vernachlässige nun alle Kanten dieses Unterkreises.

iii. Am ersten Eckpunkt des ersten Unterkreises, dessen Grad größer 0 ist, lässt man nun einen weiteren Unterkreis entstehen, der wiederum ein Eulerkreis ist.

iv. Erstelle so viele Unterkreise, bis alle Kanten von einem Unterkreis durchlaufen wurden.

v. Nun erhält man den Eulerkreis, indem man mit dem ersten Unterkreis beginnt und bei jedem Schnittpunkt mit einem anderen Unterkreis, den letzteren einfügt, und danach den ersten Unterkreis wieder bis zu einem weiteren Schnittpunkt oder dem Endpunkt fortsetzt.

Definition des Algoritmus aus der Aufgabenstellung übernommen.

Implementierung

Der Eulerkreis sowie die Unterkreise werden als Liste von Elementen abgebildet, in der immer abwechselnd ein Knoten und eine Kante eingetragen werden. Sie werden als leere Listen initialisiert.

In Initialisierungsphase des Algorithmus prüfen wir, ob es Knoten gibt, die einen ungeraden Knotengrad haben. Ist dies der Fall werfen wir eine Exception, da eine grundlegende Bedingung für die Existenz eines Eulerkreises im Graph verletzt ist. Wir werfen ebenfalls eine Exception, wenn es Knoten gibt, die mit keiner Kante verbunden sind. Ist dies der Fall, ist die Bedingungverletzt, dass es sich um einen zusammenhängenden Graphen handelt.

Andernfalls suchen wir einen beliebigen Knoten aus dem Graphen und nehmen diesen als Ausgangsknoten. Diesen fügen wir in den Unterkreis ein. Danach suchen wir so lange angrenzende Knoten und die verbindenden Kanten und fügen diese in den Unterkreis ein, bis ein Kreis entstanden ist, bzw., die letzte Kante auf den Anfangsknoten zeigt. Es werden nur Kanten eingefügt, die nocht nicht markiert wurden. Jede Kante, die in den Unterkreis eingetragen wurde wird markiert. Wenn der Unterkreis fertig ist, wird dieser, in den noch leeren Eulerkreis eingefügt.

Da dieser erste Unterkreis wahrscheinlich noch nicht der ganze Eulerkreis ist, iterieren wir über die Elemente des bisherigen Eulerkreises und suchen nach Knoten, an die noch nicht markierte Kanten angrenzen. Wird ein solcher Knoten gefunden wird ein weiterer Unterkreis, nach dem oben

beschriebenen Schema gebildet. Dabei wird der Index des Eulerkreisliste gemerkt, an dem der Unterkreis beginnt. Der vollständige Unterkreis wird an diesem Index in den Eulerkreis eingefügt. Danach wird im Eulerkreis wieder nach einem Knoten gesucht, der mit unmarkierten Kanten verbunden ist und die Prozedur wird wiederholt. Wird irgendwann kein solcher Knoten mehr gefunden, ist der Algorithmus beendet.

Tests

Wir haben die Tests vor dem eigentlichen Algorithmus implementiert. Dadurch waren wir gezwugen, uns schon im Vorfeld über mögliche Probleme und entscheidungen in der Implementierung Gedanken zu machen, was den Prozess nicht ganz einfach macht. Allerdings was dieses vorgehen später, bei der eigentlichen Implementierung von Vorteil, da man sich gut nach den Tests richten konnte. Allerdings ist es uns nicht gelungen, jedes kleine Detail im Vorfeld durch einen Test abzudecken.

Folgende Testfälle haben wir definiert:
Korrekter Graph → Es Soll ein Eulerkreis gefunden werden. Dieser muss alle Kante des Graphen genau einmal enthalten, und muss immer abwechselns aus einem Knoten und einer Kante bestehen.
Graph der Knoten mit ungeraden Grad enthält → Es Soll eine Exception geworfen werden
Graph der Knoten mit Knotengrad 0 enthält → Es soll Exception geworfen werden
100 randomisierte Graphen → Für jeden Graphen soll ein Eulerkreis gefunden werden, der alle Kanten des Graphen genau einmal enthält.

Fleury Algorithmus

Der Fleury-Algorithmus wird verwendet, um den EulerTour (Euler Zyklus) aus einem gegebenen Graphen anzuzeigen. Bei diesem Algorithmus wird ausgehend von einer Kante versucht, andere benachbarte Knoten zu verschieben, indem die vorherigen Kanten entfernt werden. Mit diesem Trick wird es in jedem Schritt einfacher, den Euler-Pfad oder die Euler-Zyklus zu finden.

Algorithmus

Gegeben sei ein eulerscher Graph $G = (V, E)$.
Ausgabe ist die Eulertour $W_{|E|}$.

Schritt 1: Man wähle einen beliebigen Knoten v_0 in G und setze $W_0 = v_0$.

Schritt 2: Wenn der Kantenzug $W_i = v_0 e_1 v_1 \dots e_i v_i$ gewählt worden ist (so daß alle $e_1 \dots e_i$ unterschiedlich sind), wähle man eine von $e_1 \dots e_i$ verschiedene Kante e_{i+1}, so daß

1. e_{i+1} inzident mit v_i ist und
2. ausgenommen, es gibt keine Alternative, e_{i+1} keine Schnittkante des Teilgraphen $G \setminus \{e_1, \dots, e_i\}$ ist.

Schritt 3: Man beende den Algorithmus, wenn W_i jede Kante von G beinhaltet. Andernfalls ist Schritt 2 zu wiederholen.

Implementierung

Wir finden zuerst den Startpunkt, der ein zufälliger Vertex u sein kann und speichern ihn in der Variablen currentNode. Um einen zufälligen Knoten zu erhalten, erhalten wir den Knoten mit einem zufälligen Index im Diagramm, solange der Index kleiner als die Anzahl der Knoten im Diagramm ist.
Wir speichern den Euler-Kreis in einer LinkedList und geben dann die Euler-Tour zurück, beginnend mit dem Startknoten u in der Methode getEulerCircle ().
Alle Kanten im Diagramm werden in der Liste EdgeList gespeichert.
Bevor wir den Fleury-Algorithmus ausführen, müssen wir die Bedingungen des Graphen überprüfen. Es muss diese Dinge erfüllen: Der gesamte Knotengrad muss gerade sein und das Diagramm ist ungerichtet, der aktuelle Knoten ist nicht null und die Kantenliste ist nicht leer. Die Eigenschaften einer EulerTour wird in der Klasse EulerCycleProperties geschrieben.

Der Algorithmus wird in folgenden Schritten programmiert:
Der Algorithmus fügt eine anfänglich leere Kantensequenz hinzu und erstellt einen Eulerkreis

1. Wählen Sie einen beliebigen Knoten als aktuellen Knoten aus.
2. Wählen Sie die Kante mit dem aktuellen Knoten in Presidents. In diesem Fall müssen zuerst Kanten ausgewählt werden, die keine überbrückten Kanten im Diagramm sind.
3. Löschen Sie die ausgewählte Kante im Diagramm und fügen Sie sie dem Eulerkreis (der Liste) hinzu.
4. Wählen Sie den anderen Knoten der ausgewählten Kante als neuen aktuellen Knoten.
5. Wenn das Diagramm Kanten enthält, fahren Sie mit Schritt 2 fort.

Eine Kante in einem ungerichteten verbundenen Diagramm ist eine Brücke, wenn durch Entfernen die Verbindung zum Diagramm getrennt wird.

Die Methode isBridge () dient dazu, die Brücken im Diagramm nacheinander zu suchen, um alle Kanten zu entfernen und festzustellen, ob das Entfernen einer Kante zu einem nicht verbundenen Diagramm führt.

Die Methode wird in folgenden Schritten programmiert:

Für jede Kante:

1. Kante vom Graphen entfernen
2. Überprüfen Sie, ob der Graph weiterhin verbunden ist. (Wir verwenden BFS, um zu überprüfen, ob wir den BFS-Pfad von finden können der aktuelle Knoten zurück zum Startknoten
3. Fügen Sie dem Diagramm eine Kante hinzu.

Test

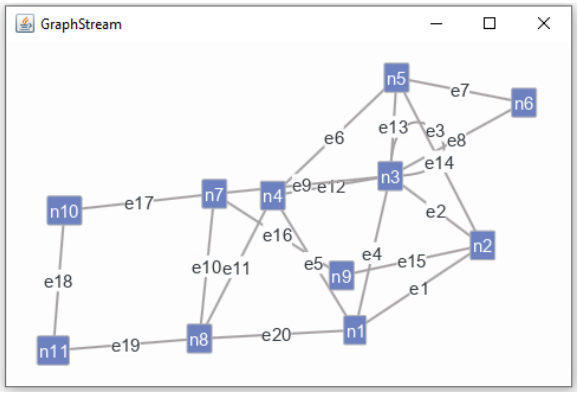
Wir implementieren folgende Tests:

Negative Test: Wir prüfen die Eingenschaften von Euler Zyklus in dem Graph.
Positive Test: Die Anzahl von Kanten in der Kreis ist gleich als die Anzahl von Kanten in der Graph

Eulergraph Generator

Der Generator erstellt Knoten und Kanten und stellt sicher, dass die letzte Kante zum ersten Knoten führt.
In einer Hashmap werden alle Knoten des generierten Graphen geführt. Key: Knoten, Value: Liste von Knoten, wo eine Kante gezogen werden kann.

```
SUCCESS Graph generated in 47 Millisec
[Key]:n8, [Value]:[n2, n6, n5, n3, n8, n9, n10]
[Key]:n9, [Value]:[n8, n6, n4, n5, n3, n1, n9, n10, n11]
[Key]:n10, [Value]:[n8, n9, n2, n6, n4, n5, n3, n1, n10]
[Key]:n7, [Value]:[n2, n6, n4, n5, n1, n7, n11]
[Key]:n2, [Value]:[n2, n4, n6, n7, n8, n10, n11]
[Key]:n6, [Value]:[n2, n4, n1, n6, n7, n8, n9, n10, n11]
[Key]:n4, [Value]:[n2, n4, n6, n7, n9, n10, n11]
[Key]:n5, [Value]:[n1, n5, n7, n8, n9, n10, n11]
[Key]:n3, [Value]:[n8, n9, n10, n11]
[Key]:n1, [Value]:[n5, n6, n7, n9, n10, n11]
[Key]:n11, [Value]:[n9, n7, n2, n6, n4, n5, n3, n1, n11]
```



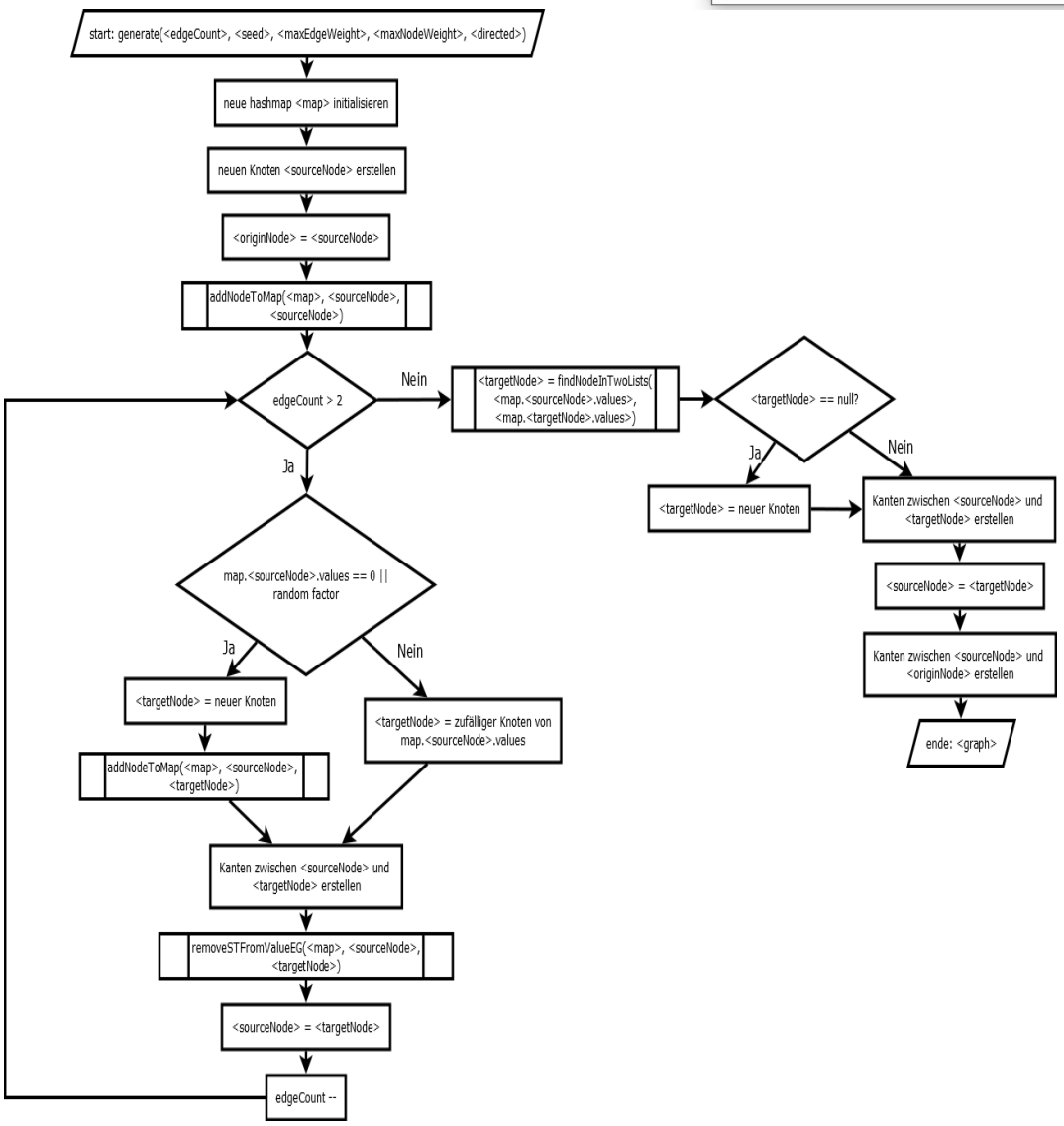


Abbildung 1Entwurf EulerGraph Generator