Die Graphschaft Schilda

Felix Möhler und Julian Thiele

Felix Möhler und Julian Thiele

Inhaltsverzeichnis

1. Die Graphschaft Schilda

1.1 Abstract

Dieses Dokument ist die Dokumentation des Projektes "Graphschaft Schilda" für das Modul Programmiertechnik III an der TH Aschaffenburg.

Die Graphschaft Schilda ist ein beschauliches Örtchen irgendwo im Nichts. Lange Zeit blieb diese Graphschaft unbehelligt vom Fortschritt, nichts tat sich in dem Örtchen. Eines Tages jedoch machte sich dort plötzlich das Gerücht breit, dass fernab der Graphschaft intelligente Menschen leben, die (fast) alle Probleme der Welt mit mächtigen Algorithmen lösen könnten. Die Bürger der Graphschaft machten sich also auf den Weg um diese intelligenten Menschen mit der Lösung ihrer Probleme zu beauftragen....

1.2 Aufgabenstellung

Entwickeln Sie ein Planungstool, dass der Graphschaft Schilda bei der Lösung ihrer Probleme hilft.

1. Analysieren Sie jedes der Probleme: Welche Daten sollen verarbeitet werden? Was sind die Eingaben? Was die Ausgaben? Welcher Algorithmus eignet sich? Welche Datenstruktur eignet sich?
2. Implementieren Sie den Algorithmus (in Java), so dass bei Eingabe der entsprechenden Daten die gewünschte Ausgabe berechnet und ausgegeben wird.
3. Geben Sie für jeden implementierten Algorithmus die Laufzeit an. Da Sie sich nun schon so viel Mühe mit dem Tool geben, wollen Sie das Tool natürlich auch an andere Gemeinden verkaufen. Die Eingaben sollen dafür generisch, d.h., für neue Orte, Feiern und Planungen anpassbar sein. Sie können diese Aufgabe ein 2er oder 3er Teams lösen. Bitte geben Sie dann die Arbeitsteilung im Dokument mit an. Die 15minütige Einzelprüfung wird auf die Projektaufgabe eingehen.

1.3 Das Team

- Felix Möhler - GitHub
- Julian Thiele - GitHub

1.4 Auftraggeber

Prof. Barbara Sprick - Professorin für Praktische Informatik bei TH Aschaffenburg

2. Problem 1 - "Straßen müssen her!"

Lange Zeit gab es in der Graphschaft Schilda einen Reformstau, kein Geld floss mehr in die Infrastruktur. Wie es kommen musste, wurde der Zustand der Stadt zusehends schlechter, bis die Bürger der Graphschaft den Aufbau Ihrer Stadt nun endlich selbst in die Hand nahmen. Zunächst einmal sollen neue Straßen gebaut werden. Zur Zeit gibt es nur einige schlammige Wege zwischen den Häusern. Diese sollen nun gepflastert werden, so dass von jedem Haus jedes andere Haus erreichbar ist. Da die Bürger der Stadt arm sind, soll der Straßenbau insgesamt möglichst wenig kosten. Die Bürger haben bereits einen Plan mit möglichen Wegen erstellt. Ihre Aufgabe ist nun, das kostengünstigste Wegenetz zu berechnen, so dass alle Häuser miteinander verbunden sind (nehmen Sie dabei pro Pflasterstein Kosten von 1 an):

2.1 Modellierung des Problems

Das Problem lässt sich als Graphenmodell mit ungerichteten Kanten darstellen. Jedes Haus ist ein Knoten, die Straßen sind die Kanten. Die Kosten der Kanten sind die Kosten für die Pflastersteine.

Um den Graph zu modellieren werden die Java-Bibliotheken `JGraphT` und `JGraphX` verwendet. Mit `JGraphT` wird der Graph als Datenstruktur modelliert. Mit `JGraphX` wird der Graph als Grafik dargestellt und auf dem Bildschirm dargestellt.

2.2 Die Eingabe

Die Eingabe besteht aus einem Graphen, der aus Kanten und Knoten besteht. Diese werden aus einer `.json` Datei gelesen und in eine Insanz der Klasse `GraphData.java` geladen. Diese Insanz dient als Basis für die Berechnung des günstigsten Weges.

```
{
  "directed_edges": false,
  "vertices": [
    {
      "label": "Wasserwerk"
    }, {
      "label": "Thoma"
    }
    ...
  ],
  "edges": [
    {
      "source": "Wasserwerk",
      "target": "Thoma",
      "weight": 15
    },
    ...
  ]
}
```

2.3 Die Ausgabe

Die Ausgabe wird als Graph in einem Fenster dargestellt. Das Fenster besteht aus zwei Hälften. Auf der linken Seite wird der Eingabegraph dargestellt. Auf der rechten Seite wird der berechnete Graph dargestellt.

Problem1

2.4 Der Alrogithmus

TODO Beschreibung MST mit Prim

2.5 Die Laufzeit des Algorithmus

TODO Laufzeitberechnung

2.6 Die Implementation des Algorithmus

```
// Initialisiere alle Knoten mit ∞, setze den Vorgänger auf null
for (GraphVertex v : vertices) {
  v.setValue(Integer.MAX_VALUE);
  v.setPredecessor(null);
}

// Starte mit beliebigem Startknoten, Startknoten bekommt den Wert 0
GraphVertex start = vertices.get(6);
start.setValue(0);

// Speichere alle Knoten in einer geeigneten Datenstruktur Q
// -> Prioritätswarteschlange
PriorityQueue<GraphVertex> queue = new PriorityQueue<GraphVertex>(vertices.size(), new VertexComparator());
queue.addAll(vertices);

// Solange es noch Knoten in Q gibt...
while (!queue.isEmpty()) {
  // Wähle den Knoten aus Q mit dem kleinsten Schlüssel (v)
  GraphVertex vertex = queue.poll();

  // Speichere alle Nachbarn von v in neighbours
  ArrayList<GraphVertex> neighbors = GraphData.getNeighbors(vertex, vertices, edges);

  for (GraphVertex n : neighbors) {
    // Finde Kante zwischen v und n
    for (GraphEdge edge : GraphData.getEdgesBetweenTwoVertices(vertex, n, edges)) {
      // Wenn der Wert der Kante kleiner ist als der Wert des Knotens
      // prüfe ob der Knoten noch in Q ist
      if (edge.getWeight() < n.getValue() && queue.contains(n)) {
        // Speichere v als vorgänger von n und passe wert von n an
        n.setValue((int) edge.getWeight());
        n.setPredecessor(vertex);
        // Aktualisiere die Prioritätswarteschlange
        queue.remove(n);
        queue.add(n);
      }
    }
  }
}
```

3. Problem 2 - "Wasserversorgung"

3.1 Modellierung des Problems

3.2 Die Eingabe

3.3 Die Ausgabe

3.4 Der Alrogithmus

3.5 Die Laufzeit des Algorithmus

3.6 Die Implementation des Algorithmus

4. Problem 3 - "Stromversorgung"

4.1 Modellierung des Problems

4.2 Die Eingabe

4.3 Die Ausgabe

4.4 Der Alrogithmus

4.5 Die Laufzeit des Algorithmus

4.6 Die Implementation des Algorithmus

5. Problem 4 - "Historische Funde"

5.1 Modellierung des Problems

5.2 Die Eingabe

5.3 Die Ausgabe

5.4 Der Alrogithmus

5.5 Die Laufzeit des Algorithmus

5.6 Die Implementation des Algorithmus

6. Problem 5 - "Die Festhochzeit - das Verteilen der Einladungen"

6.1 Modellierung des Problems

6.2 Die Eingabe

6.3 Die Ausgabe

6.4 Der Alrogithmus

6.5 Die Laufzeit des Algorithmus

6.6 Die Implementation des Algorithmus

7. Problem 6 - "Wohin nur mit den Gästen?"

7.1 Modellierung des Problems

7.2 Die Eingabe

7.3 Die Ausgabe

7.4 Der Alrogithmus

7.5 Die Laufzeit des Algorithmus

7.6 Die Implementation des Algorithmus

8. Problem 7 - "Es gibt viel zu tun! Wer macht's"

8.1 Modellierung des Problems

8.2 Die Eingabe

8.3 Die Ausgabe

8.4 Der Alrogithmus

8.5 Die Laufzeit des Algorithmus

8.6 Die Implementation des Algorithmus