Praktikum Algorithmen

WS 2012/13

TECHNISCHE UNIVERSITÄT DARMSTADT

Readme

Einleitung

Dieses Dokument gibt einen Einblick in den Umgang mit den Programmen, die im Algorithmen-Praktikum zur Anwendung kommen. Bei diesen Programmen handelt es sich um

a) OSM Railway Graph Import

b) Shortest Routes Finder

c) Simple Routes Filter

d) RailViz

e) Web-Frontend von MOTIS (RaNDM)

Im Folgenden wird beschrieben, was die einzelnen Programme machen, wie man sie startet und welche Anforderungen sie stellen.

OSM Railway Graph Import

Das Programm *OSM Railway Graph Import* verarbeitet einen Datensatz von OpenStreetMap, indem es daraus einen Schienennetz-Graph extrahiert und diesen in eine MongoDB[1] Datenbank speichert. Das Ausführen des Programms erfordert daher, dass MongoDB installiert und gestartet ist. Wie die MongoDB Datenbank gestartet wird, hängt vom System und der Installation ab und kann auf der Homepage von MongoDB nachgelesen werden. Datensätze von OpenStreetMap sind verfügbar unter `http://download.geofabrik.de/openstreetmap/`. Von Bedeutung ist prinzipiell nur der Datensatz, der Deutschland umfasst (`http://download.geofabrik.de/openstreetmap/europe/germany.osm.bz2`). Allerdings kann es hilfreich sein, zu Testzwecken mit deutlich kleineren Datensätzen zu arbeiten (etwa mit Mittelfranken), da dies zu einer erheblich geringeren Ausführungsdauer führt.
Das Programm *OSM Railway Graph Import* lässt sich durch den Befehl

```
java -jar -Xms256m -Xmx1500m osm_railway_graph_import.jar germany.osm localhost 27017
```

in der Konsole starten. Das Bereitstellen von mehr Arbeitsspeicher kann die Ausführungszeit beschleunigen (→ Argument `-Xmx1500m` verändern). Das Argument `germany.osm` verweist auf den Datensatz von OpenStreetMap. Befindet sich dieser in einem anderen Verzeichnis als das Programm oder soll ein anderer Datensatz eingelesen werden, muss das Argument entsprechend angepasst werden. Die beiden letzten Argumente beschreiben Host und Port der Datenbank. Standardmäßig ist diese lokal über Port 27017 zu erreichen. Sollte es passieren, dass `localhost` nicht korrekt übersetzt wird und das Programm infolgedessen abstürzt, so kann auch `127.0.0.1` anstelle von `localhost` verwendet werden. Wenn die Datenbank nicht lokal und/oder über einen anderen Port gestartet wird, müssen die beiden letzten Argumente entsprechend angepasst werden.
Befindet sich die Datei *germany.osm* im gleichen Verzeichnis wie das Programm und ist die Datenbank lokal über Port 27017 zu erreichen, so lässt sich das Programm auf Unix-Systemen auch mit dem Befehl

```
sh osm_railway_graph_import.sh
```

in der Konsole starten. An dieser Stelle sei noch erwähnt, dass bei Verwendung eines Rechners mit 2.2 GHz Intel Core 2 Duo Prozessor und 2GB RAM das Ausführen des Programmes knapp 40 Minuten dauerte.
Der in die Datenbank gespeicherte Schienennetz-Graph befindet sich dann in der *Collection* “railway_graph” und ist eine Ansammlung von Knoten. Jeder dieser Knoten kennt seine Nachbarknoten und die Entfernung zu diesen. Ein solcher Knoten hat also die folgende Gestalt:

[1] `http://www.mongodb.org/`

```
{
    "_id" : 1692531820,
    "loc" : [
        8.4906362,
        49.1566117
    ],
    "neighbours" : [
        {
            "id" : 1692531818,
            "distance" : 5
        },
        {
            "id" : 1692531823,
            "distance" : 11
        },
        {
            "id" : 1692531824,
            "distance" : 20
        }
    ]
}
```

Über das Feld `_id` lässt sich ein Knoten eindeutig identifizieren. Das Feld `loc` enthält die Koordinaten des Knotens (Längengrad, gefolgt von Breitengrad). Das Feld `neighbours` enthält die Nachbarn des Knotens (deren Bezeichner sowie die Distanz in Metern zum betrachteten Knoten).

Shortest Routes Finder

Das Programm *Shortest Routes Finder* importiert den Schienennetz-Graph aus der Datenbank und sucht darauf kürzeste Wege. Die Dateien *sgrv.csv* und *Stations.txt* enthalten die Information, welche Routen gesucht werden sollen und müssen daher im gleichen Verzeichnis liegen wie das Programm. Um Zugriff zur MongoDB Datenbank zu haben, muss diese zuvor gestartet werden. Das Programm kann dann mit dem Befehl

```
java -jar -Xmx1500m shortest_routes_finder.jar railway_graph 16 localhost 27017
```

gestartet werden. Auch hier gilt, dass das Bereitstellen von mehr Arbeitsspeicher die Ausführungszeit beschleunigen kann (→ Argument `-Xmx1500m` verändern). Die beiden letzten Argumente beschreiben erneut Host und Port der MongoDB Datenbank. Das Argument `railway_graph` ist der Name der *Collection* in der Datenbank, in der sich der Schienennetz-Graph befindet, auf welchem kürzeste Wege gesucht werden sollen. Wird für jedes Zoom-Level ein eigener Schienennetz-Graph erstellt, muss das Argument bei der Routensuche auf einem der neu erstellten Graphen selbstverständlich angepasst werden. Die Zahl 16 gibt an, dass die gefundenen Routen zum Zoom-Level 16 gehören und darum in der *Collection* "shortest_routes_zoom_level_16" gespeichert werden sollen. Das Zoom-Level entspricht hier dem Zoom-Level von Google Maps. Letztlich benötigt RailViz (siehe Abschnitt RailViz) solche *Collections* für die Zoom-Stufen 8 bis 16.
Das Programm *Shortest Routes Finder* kann auf Unix-Systemen auch in der Konsole mit dem Befehl

```
sh shortest_routes_finder.sh
```

gestartet werden. Auch hier gilt, dass die Ausführung des Programms etliche Minuten dauern kann (je nach CPU, Anzahl der Kerne, bereitgestellter RAM).
Die gefundenen Routen werden wie bereits erwähnt in einer *Collection* mit Name "shortest_routes_zoom_level_i" in der MongoDB Datenbank gespeichert, wobei `i` dem übergebenen Parameter (Wert zwischen 8 und 16) entspricht. Eine Route hat dabei folgende Form:

```
{
  "_id" : { "$oid" : "507b525103648dc5dc3a8e32"},
  "route_nodes" : [
    {"lon" : 8.60341, "lat" : 49.7596},
    {"lon" : 8.6035303, "lat" : 49.7599823},
    {"lon" : 8.603607, "lat" : 49.7602789},
    {"lon" : 8.6037285, "lat" : 49.7607052},
    ...
    {"lon" : 8.62596, "lat" : 49.8142}
  ],
  "source" : {"lon" : 8.60341, "lat" : 49.7596},
  "target" : {"lon" : 8.62596, "lat" : 49.8142}
}
```

Das Feld `_id` wird beim Einfügen der Routen in die Datenbank automatisch generiert und erlaubt die eindeutige Identifizierung der Route. Das Feld `route_nodes` ist eine Sequenz aus Koordinatenpaaren und beschreibt die Knoten der Route. Während `lon` den Längengrad eines solchen Knotens bezeichnet, steht `lat` für den Breitengrad des Knotens. Die Felder `source` und `target` beschreiben die beiden Endpunkte der Route und erlauben damit das Abfragen einer Route bei gegebenen Endpunkten.

Simple Routes Filter

Das Programm *Simple Routes Filter* wird dafür verwendet, aus den vollständigen Routen für Zoom-Stufe 16 Routen für die Zoom-Stufen 8-15 zu generieren, indem Punkte weggelassen werden. Dabei wird ein sehr simpler Ansatz verwendet, der die Relevanz von Knoten nicht berücksichtigt. Das Programm kann insofern als Platzhalter interpretiert werden - es soll in diesem Praktikum durch einen besseren Ansatz ersetzt werden. Zum besseren Verständnis befindet sich der Sourcecode des Programms in der Datei *simple_routes_filter.zip* und kann als Projekt in Eclipse importiert werden.
Um das Programm zu starten, bedarf es des Befehls

```
java -jar -Xmx1500m simple_routes_filter.jar localhost 27017
```

in der Konsole. Die beiden Argumente `localhost` und `27017` beschreiben Host und Port, über welche die MongoDB Datenbank erreicht werden kann.
Das Programm kann (mit den obigen Standardargumenten) auf Unix-Systemen alternativ auch per Befehl

```
sh simple_routes_filter.sh
```

in der Konsole gestartet werden.

RailViz

RailViz ist ein Zugmonitor, in welchem unter Anderem die genannten Routen visualisiert werden. Zu diesem Zweck importiert *RailViz* die Routen aus der MongoDB Datenbank. Wie bereits erwähnt, bedarf es dafür *Collections* in der MongoDB Datenbank mit den Namen “shortest_routes_zoom_level_i” für alle i von 8 bis einschließlich 16. *RailViz* wird unter Unix-Systemen über den Befehl

```
railviz_2.0_simulation.sh
```

in der Konsole gestartet. Auf diese Weise wird folgender Befehl ausgeführt:

```
java -Xmx1536m -classpath railviz_2.0.jar railViz/RailViz -p conf/config.xml -s -dr
```

RailViz liest die benötigten Datensätze nicht bei jedem Programmstart ein, sondern serialisiert relevante Klassen und speichert diese Serialisierung in eine eigene Datei im Verzeichnis *data/november2/serialized/*. Existiert eine solche Datei, wird diese bei einem Programmstart geladen, was gegenüber dem Einlesen der benötigen Datensätze zu einer signifikanten Beschleunigung des Programmstartes führt. Haben sich die Routen in der Datenbank geändert, muss das Verzeichnis *data/november2/serialized/* entfernt werden oder aber das Argument `-sr` wird in letztgenanntem Konsolenbefehl ergänzt (führt zu einem Überschreiben der serialisierten Daten).
Um einen Performanzgewinn zu erreichen, empfiehlt es sich, *RailViz* mehr Arbeitsspeicher zur Verfügung zu stellen (→ Argument `-Xmx1536m` verändern). Ist die Datenbank nicht lokal über Port 27017 zu erreichen, muss die folgende Zeile in der Konfigurationsdatei *config.xml* im Ordner *conf* angepasst werden:

```
<mongoDatabase host="localhost" port="27017" />
```

Web-Frontend von MOTIS (RaNDM)

Das Web-Frontend von MOTIS erlaubt den Zugriff auf die Oberfläche von *RailViz*. Um das Web-Frontend verwenden zu können, müssen Ruby als auch Rails installiert sein. Anschließend muss im Ordner *code* der Befehl

```
bundle install
```

in der Konsole ausgeführt werden. Unter Windows muss zuvor noch die Zeile

```
gem 'therubyracer'
```

aus der Datei *gemfile* im Verzeichnis *code* entfernt werden.
Das Web-Frontend lässt sich dann per Befehl

```
rails s
```

im Ordner *code* in der Konsole starten. Die Oberfläche von *RailViz* ist anschließend im Browser unter der URL

```
localhost:3000/
```

zu erreichen. Um den Zugmonitor nutzen zu können, muss selbstverständlich *RailViz* ausgeführt werden.