Programmierprojekt:
Ein webbasierter Routenplaner

Tobias Rupp

(FMI/ALG)

http://www.fmi.uni-stuttgart.de/alg

Aufgaben und Ziele

Implementierung eines webbasierten Routenplaners

- Backend: Graphrepräsentation, Routenberechnung;läuft auf “Server”
- GUI: Web-Client (Javascript, Leaflet)
- Webserverkomponente: Verbindung zwischen Backend und GUI

Ziele / Was lernen Sie?

- Umsetzung eines nicht-trivialen Projektes in einer Kleingruppe
- Arbeiten mit nicht ganz kleinen Datenmengen

Ablauf

- Aufteilung des Projektes in 3 Phasen entsprechend den Komponenten
- Abnahme nach Ende jeder Phase
- Fortführung des Projektes nur nach erfolgreicher Abnahme

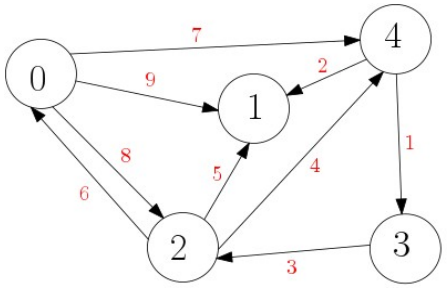
Phase I: Backend: Routenplanung
- Effiziente Darstellung von Straßennetzwerken im Speicher
- Routenplanung mittels Dijkstra
- Deadline: **20.06.2019**

Phase II: Backend: Lokalisierung
- Nächste-Nachbarn-Suche
- Effizient mittels k-d tree
- Deadline: **20.07.2019**

Phase III: GUI/Webclient und Webserver
- Kartendarstellung in Browser
- Interaktion mit Benutzer
- Erstellung und Verknüpfung von Server und Client
- Deadline: **31.08.2019**

Phase I: Backend – Graphdarstellung

- Auf http://www.fmi.uni-stuttgart.de/alg/research/stuff/ finden sich Graphen verschiedenster Größen
- 1. Aufgabe von Phase I:
  - Implementieren Sie eine Klasse, welche derartige Graphen effizient im Speicher darstellt
  - Ihre Klasse sollte auf der OffsetArray-Darstellung basieren um schnell ausgehende Kanten zu einem Knoten zurückliefern zu können
  - Vermeiden Sie Java-Objekte (!), nutzen Sie ausschließlich elementare Datentypen (int, double)
  - Ein Rechner mit 8GB sollte problemlos den Deutschlandgraph (n=20Mio, m=40Mio) im RAM halten können



node	offset
0	0
1	3
2	3
3	6
4	7
5	9

edge idx	src	trg	weight
0	0	1	9
1	0	2	8
2	0	4	7
3	2	0	6
4	2	1	5
5	2	4	4
6	3	2	3
7	4	1	2
8	4	3	1

Phase I: Backend – Dijkstras Algorithmus

- 2. Aufgabe von Phase I:
  - Implementieren Sie Dijkstras Algorithmus
  - Hinweise:
    - Einfachste Variante: PriorityQueue aus der Standardbibliothek mit lazy inserts, d.h. wenn ein Knoten schneller erreichbar wird, wird er mit den neuen Kosten nochmals der PriorityQueue hinzugefügt. Duplikate werden später beim Poppen aus der PriorityQueue einfach ignoriert.
    - Vermeiden sie die Nutzung von (Hash)maps
  - Auf dem Deutschlandgraph (n=25Mio, m=50Mio) sollte eine Anfrage wenige Sekunden dauern

Phase II: Backend – Nächster Knoten simpel

- 1. Aufgabe von Phase II:
  - Geg. Eine Position Lon./Lat., finde nächsten Knoten
  - Finde Sie den Knoten zunächst mit einer simplen Iteration über alle Knoten.
  - Nehmen Sie einfachheitshalber Lon./Lat. als x und y im euklidischen Raum.
  - Hinweis:
    - Der nächste Knoten ist nicht immer eindeutig.

Phase II: Backend – Nächster Knoten mit k-d tree

- 2. Aufgabe von Phase II:
  - Gleiche Aufgabenstellung wie in 1. Aufgabe
  - Implementieren Sie dazu einen k-d-Tree (k=2) der das Gebiet partitioniert und eine effizientere Suche ermöglicht.
  - Hinweis:
    - Der nächste Knoten und die gegebene Position können in unterschiedlichen Partitionen liegen!! Deswegen muss eine Art Tiefensuche durchgeführt werden, die nach dem Abstieg in einen Teilbaum den anderen ggf. prunen kann.

Phase II: Backend – Nächster Knoten Unittest

- 3. Aufgabe von Phase II:
  - Schreiben Sie einen Unittest der die Ergebnisse ihrer k-d Tree Implementierung aus 3.2. mit dem simplen Iterieren aus 3.1 verifiziert.
  - Der Test soll 100 zufällige Koordinaten in der minimalen Bounding Box des Graphen samplen. Inbesondere sollen nicht einfach die Koordinaten von den Knoten für den Test genommen werden.

Phase III: GUI/Webclient

- 1. Aufgabe von Phase III:
  - Machen Sie sich mit der JavaScript Bibliothek Leaflet (http://www.leafletjs.com) vertraut
  - Erstellen Sie zunächst eine einfache HTML-Seite, welche:
    - Mit OpenStreetMap als Tile-Provider einen Kartenausschnitt darstellt, der Zoom- und Scrolloperationen unterstützt
    - Auf Mausclick die entsprechende Position in Längen- und Breitengrad in einem Anzeigeelement der Seite darstellt
  - Finden Sie heraus, wie man geometrische Objekte (Pfade) im GeoJSON-Format als Layer über die Karte legt

Phase III: Anbindung Webclient <->Backend

- 2. Aufgabe von Phase III:
  - Erstellen Sie einen einfachen Webserver, welcher
    - den HTML-Code bereitstellt, der von einem Webbrowser (Client) ausgeführt werden kann.
    - Anfragen (Ajax-Requests) des Clients entgegen nimmt, die entsprechenden Methoden im Backend aufruft und deren Ergebnis an den Client zurückschickt.
  - Mindestfunktionalität:
    - „Nächster Knoten“: Gegeben Längen-&Breitengrad: Rückgabe des nächsten Knotens (ID, Längen-&Breitengrad)
    - „Routenplanung“: Gegeben KnotenIDs von Start und Ziel: Berechne Route, und gebe Pfad zur Visualisierung zurück

Empfehlungen

- Nutzen Sie eine IDE (Eclipse, o.Ä.) und deren Möglichkeiten wie z.B. Profiler, Debugger, Autoformatting ...
- Nutzen Sie ein automatisiertes Buildsystem wie Maven.
- Machen Sie die Installation, die Benutzung ihres Programms und die Verifikation der Anforderungen möglichst einfach für einen Benutzer.
- Testen Sie ihr Programm auf dem Zielsystem, einer VM in der bwCloud.

Stolpersteine

- Encoding: US-ASCII mit Umlauten kann Kompilieren verhindern, am besten in VM auf bwCloud testen
- Bei OutOfMemoryException: JVM-Parameter für Arbeitsspeicher: z.B. java -Xmx8g
- Bei multidimensionalen Arrays macht es einen Unterschied ob Sie als new int[3][50000000] oder new int[50000000][3] allokiert werden.