1 Lernziele

Es soll eine Problemstellung aus dem Bereich der effizienten Algorithmen eigenständig in einer Gruppe bearbeitet werden. Dazu ist eine Einarbeitung in das Themengebiet und das Erstellen eines Projektplans notwendig. Die Algorithmen und Datenstrukturen sollen implementiert und in einem konkreten Anwendungsfall analysiert werden. Dabei ist das Problem mit Fachwissen unter Anwendung geeigneter Methoden und Verfahren entsprechend des aktuellen Stands der Technik zu lösen, das Projekt zu planen, durchzuführen, abzuschließen und zu dokumentieren.

Allgemein Täglich werden optimale Routen in Verkehrsnetzen berechnet. In diesem Projekt sollen verschiedene Teilaspekte der Routenplanung betrachtet werden. Die zu erstellende Software ist prinzipiell auch auf mobilen Geräten mit sehr begrenzter Hardware lauffähig.

Open-Street-Map Frei verfügbare Karten, die zur Routenplanung genutzt werden können, findet man im Internet zum Beispiel bei Open-Street-Map[1]. In diesem Projekt soll es darum gehen, solche Karten in den Computer einzulesen, aufzubereiten, Routen zu planen, diese Routen in geeigneter Form auszugeben sowie die Algorithmen und Datenstrukturen zu analysieren.

External Memory Data Structures Um die Routenplanung auch auf sehr großen Karten und auch auf Computern mit beschränkten Hardware-Ressourcen durchführen zu können, sind Datenstrukturen notwendig, die externen Speicher wie Festplatten oder SD-Karten nutzen. Da die Routenplanung effizient sein soll, ist die Software in C++ zu implementieren. Bibliotheken wie die Boost Graph Library[2] oder die STxxL[3] (Standard Template Library for Extra Large Data Sets) können und sollen in diesem Projekt genutzt werden.

Algorithmen für moderne Hardware Moderne Hardwarearchitekturen bauen auf einer Speicherhierarchie auf, die von Prozessorregistern über First-, Second- und Third-Level-Cache, RAM (also Hauptspeicher) bis zur Festplatte oder noch langsameren Massenspeichern wie Bandlaufwerken reicht. Daten werden dann jeweils in Blöcken einer festen Größe aus nachgeordneten Speicherebenen geladen. Da die Zugriffszeiten auf die Speicherebenen jeweils um Größenordnungen auseinanderliegen können, müssen die Zugriffe möglichst effizient gestaltet werden. Hierarchisches Speichermodell[4] für Intel Core i7:

- 1 cycle to read a CPU register
- 4 cycles to reach L1-Cache (32 kB + 32 kB, 64 byte per line)
- 11 cycles to reach L2-Cache (256 kB, 64 byte per line)
- 39 cycles to reach L3-Cache (2 MB per core (shared), 64 byte per line)
- ≈100 cycles to reach main memory (RAM)
- ≈10.000 of cycles to reach external memory

[1] `https://www.openstreetmap.de/`
[2] `https://www.boost.org/doc/libs/1_79_0/libs/graph/doc/index.html`
[3] `https://stxxl.org/`
[4] `https://people.freebsd.org/~lstewart/articles/cpumemory.pdf`

Wir müssen also versuchen unsere Programme so zu schreiben, dass die Daten möglichst im Cache vorhanden sind, wenn wir sie benötigen. Um die Daten für kürzeste-Wege-Algorithmen Cache-effizient nutzen zu können, müssen die Knoten des Graphen z.B. mittels geo-hashing[5] sortiert werden.

2 Routenplanung

Grundlage für die Routenplanung sind Graphen und Kürzeste-Wege-Algorithmen. Die Knoten des Graphen repräsentieren die Straßenkreuzungen, durch Kanten werden einzelne Straßenabschnitte modelliert. Um Einbahnstraßen darstellen zu können, basieren Routenplaner auf gerichteten Graphen. Außerdem besitzt jede Kante ein Gewicht, das die Länge des Straßenabschnitts modelliert.

Datenstrukturen Zunächst ist eine Datenstruktur für Graphen zu implementieren, mit der das Programm zur Routenplanung arbeiten kann. Dazu kann z.B. die „Boost Graph Library“ genutzt werden. Spezielle, für die Verwendung in Routenplanern optimierte Datenstrukturen sind bspw. in den folgenden Papieren beschrieben:

- Andrew V. Goldberg and Renato F. Werneck: Computing Point-to-Point Shortest Paths from External Memory. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments, 2005[6].
- Peter Sanders, Dominik Schultes, and Christian Vetter: Mobile Route Planning. Universität Karlsruhe, 2008[7].

Die Algorithmen zur Routenplanung müssen bei großem Kartenmaterial mit Datenstrukturen wie einer Priority-Queue oder einer Liste arbeiten, die externen Speicher nutzen.

Einlesen der OSM-Daten Die Rohdaten müssen eingelesen, aufbereitet und in die obige Datenstruktur eingefügt werden. Zunächst können mit einem Tool wie `osmfilter` die Daten von unnötigen Informationen wie Autor, Version, Änderungsdatum usw. entfernt werden. Das eigentliche Kartenmaterial kann trotz der Filterung sehr umfangreich sein, zum Beispiel hat die Datei, die die Europa-Karte[8] enthält, eine Größe von über 20 GB. Daher müssen auch beim Einlesen der Daten ggf. External-Memory-Datenstrukturen genutzt werden. Informationen zu diesem Thema findet man in dem Buch von Jeffrey Scott Vitter: Algorithms and Data Structures for External Memory[9]. Entweder müssen solche Datenstrukturen implementiert werden oder es muss untersucht werden, ob z.B. die STxxL geeignete Datenstrukturen enthält.

Vorverarbeitung Zur Vorverarbeitung gehört zum Beispiel, dass Pfade, deren innere Knoten vom Grad zwei sind, die also keine Kreuzung darstellen, zu einer einzigen Kante im Graphen reduziert werden, wobei die Länge der Kante der Länge des Pfades entspricht. Warum? Die Laufzeit von Dijkstras Algorithmus ist:

$$T \;=\; \Theta(\mathcal{V}) \cdot T_{ExtractMin} \;+\; \Theta(\mathcal{E}) \cdot T_{DecreaseKey}$$

[5] `https://en.wikipedia.org/wiki/Geohash`
[6] `https://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/GW05.pdf`
[7] `http://algo2.iti.kit.edu/schultes/hwy/mobileSubmit.pdf`
[8] `http://download.geofabrik.de/europe-latest.osm.pbf`
[9] `https://www.ittc.ku.edu/~jsv/Papers/Vit.IO_book.pdf`

Dabei bezeichnet $T_{ExtractMin}$ bzw. $T_{DecreaseKey}$ die Worst-Case-Laufzeit einer Extract-Min- bzw. Decrease-Key-Operation auf der Vorrangwarteschlange (Priority Queue). Eine Reduzierung der Anzahl $\mathcal{V}$ der Knoten und der Anzahl $\mathcal{E}$ der Kanten hat also einen großen Einfluss auf die Laufzeit, die für die Routenplanung benötigt wird.

Abbiegeinformationen Oft ist an Kreuzungen das Abbiegen in bestimmte Richtungen nicht erlaubt, was durch verschiedene Verkehrsschilder[10] ausgedrückt wird.

Solche Abbiegeinformationen müssen bei der Routenplanung berücksichtigt werden. Dazu ist zu überlegen, wie diese Informationen abgelegt werden können, ohne viele Ressourcen zu benötigen, und wie die Algorithmen bzw. die Graphen angepasst werden müssen.

Routenplanung Verschiedene Algorithmen zur Berechnung kürzester Wege wie Dijkstra, bidirektionaler Dijkstra, Bellman/Ford und der $A^\star$-Algorithmus sollen implementiert und deren Laufzeiten anhand von konkreten Beispielen verglichen werden. Dabei ist die Auswirkung der External-Memory-Datenstrukturen zu untersuchen.

Darstellung der Routen Damit die Routen auch visuell kontrolliert werden können, ist eine grafische Benutzeroberfläche zu erstellen, mittels derer auch der Start und das Ziel eingegeben werden können. Wenn die Route ermittelt wurde, ist diese Route in der Karte anzuzeigen, dazu steht eine API vom Open-Street-Map-Projekt zur Verfügung.

Teilaspekte Die folgenden Aufgaben müssen unter den Teilnehmern aufgeteilt und bearbeitet werden:

- Aufbau und Vorverarbeitung von Open-Street-Map-Daten
- Visualisierung von Routen und erstellen einer GUI
- Darstellung von Graphen sowie Berechnung von kürzesten Wegen
- Berücksichtigen von Abbiegeinformationen in Graphen und Algorithmen
- Nutzen von externen Datenstrukturen aufgrund begrenzter Hardware
- Datenorientiertes Design, Nutzen von Cache-Effekten durch Geo-Hashing
- Beschleunigung der Algorithmen durch anreichern des Netzwerks mit Zusatzinformationen in einer Vorverarbeitungsphase

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden.

[10] `https://wiki.openstreetmap.org/wiki/DE:Relation:restriction`