Themen für das Praktikum Algorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2024

- Bewerbungen auf Praktikumsthemen werden bis zum **25.04.2024** um **08:00** unter der E-Mail Adresse **lab@algo.informatik.tu-darmstadt.de** angenommen. Danach ist keine Bewerbung mehr möglich. Bewerbungen unter anderen E-Mail Adressen werden ignoriert. Jede Bewerbung muss drei Themen mit Prioritäten (Erst-, Zweit- und Drittwunsch) enthalten. Zur Bewerbung auf ein Thema schreiben Sie uns bitte eine kurze Motivation, warum Sie genau dieses Thema bearbeiten möchten sowie Ihre Vorkenntnisse.
- Das Kick-Off Treffen findet nach Vereinbarung für jedes Thema separat statt.
- Um die Integration der Praktikumsergebnisse in die quelloffenen Projekte SORO-S/O bzw. MOTIS zu ermöglichen, müssen alle Teilnehmer Ihre Abgaben unter die MIT / Apache2 Lizenz stellen. Nach dem Kick-Off senden Sie dem Betreuer des Themas eine unterschriebene Lizenzvereinbarung (MIT / Apache2) und falls nötig eine unterschriebene Vertraulichkeitserkärung für die für das Praktikum bereitgestellten Daten. Dies wird gleichzeitig als verbindliche Zusage zur Bearbeitung des Themas angesehen.
- In einem zweiwöchigen Rythmus finden regelmäßige verpflichtende Treffen statt. Die genauen Termine werden beim Kick-Off Treffen vereinbart. Bei jedem Termin stellt jeder Teilnehmer einen Foliensatz mit 3 Folien vor (max. 5 min):
  - Welche Fortschritte / Erkenntnisse wurden seit dem letzten Treffen erreicht?
  - Welche nächsten Schritte sind geplant?
  - Liste mit Fragen - bitte achten Sie darauf, dass die Bewertungskriterien unter anderem selbstständiges Arbeiten berücksichtigen. Hier sollten Sie folglich nur Fragen stellen, deren Beantwortung nicht durch Google / bzw. das Lesen von bereitgestelltem Material möglich ist.
- Die endgültige Abgabe besteht aus:
  - Code, der für dieses Praktikum geschrieben wurde (beinhaltet Code, der Auswertungen, Tests oder ähnliches durchführt).
  - Einen Praktikumsbericht, in dem Sie beschreiben, welche Probleme Sie wie gelöst haben und wieso Sie diesen konkreten Ansatz gewählt haben.

Es gelten folgende Markierungen:

- **A**: Das Thema kann als Praktikum Algorithmen (6CP) oder Vertiefungspraktikum Algorithmen (6CP) bearbeitet werden.
- **P**: Das Thema kann als Projektpraktikum (9CP) bearbeitet werden.
- **E**: Das Thema kann einzeln bearbeitet werden.
- **G**: Das Thema kann auch als Gruppe bearbeitet werden.

Der Umfang des Themas wird den Umständen entsprechen angepasst. Die Praktikumsform 'Projektpraktikum', als auch Gruppenarbeit, erhöhen selbstverständlich den Arbeitsumfang.

Der Abgabetermin des Praktikums ist der 30.09.2024.

Hinweise:

- Planen Sie bitte zusätzlichen Aufwand ein, falls die genannten Programmiersprachen (C++, Typescript, Javascript, CUDA) oder verwendeten Technologien (z.B. Git, CMake, React, etc.) neu für Sie sind.
- Betrifft alle C++ Praktika: Als Referenz für die Komplexität des Projektes, das im Praktikum erweitert wird, können Sie MOTIS[1] anschauen. Das SORO Projekt aus den Themen 2.x ist ähnlich komplex.
  Bitte überlegen Sie sich genau, ob Sie mit einer großen "real-world" Modern-C++ Code Basis klarkommen. Bisherige Erfahrungen haben gezeigt, dass Studierende den Einarbeitungsaufwand und die Komplexität der Programmiersprache unterschätzen.
  Einen Crash-Kurs in die MOTIS/SORO-typische C++-Programmierung gibt es hier: `https://www.algo.informatik.tu-darmstadt.de/dl/cpp.pdf`.
- Falls Themen in Gruppen bearbeitet werden, steigt der Aufwand linear mit der Summe der CPs der Gruppenmitglieder. Die unten genannten Themen sind auf Einzelpersonen ausgelegt. Bei einer Bearbeitung in der Gruppe (nur bei Themen mit **G** möglich!) wird der Umfang entsprechend erweitert.

[1] `https://github.com/motis-project/motis`

Thema 1: Serialisierbarer Quad-Tree/R-Tree mit `cista` *[A, P, E]*

Die `cista`[2] Library erlaubt die effiziente Serialisierung von C++ Datenstrukturen. Grundlegende Datenstrukturen wie `string`, `vector`, etc. werden von `cista` bereits reimplementiert. Aufgabe in diesem Praktikumsthema ist es, eine effiziente Geo-Lookup Datenstruktur in `cista` zu implementieren und gegen gängige Implementierungen (wie z.B. Boost Geomertry R-Tree) bezüglich Zeit zum Aufbauen, Lookup-Dauer sowie der benötigten Zeit für schreiben/lesen von der Festplatte zu benchmarken (z.B. mit Google benchmark[3]).

Programmiersprache: C++

Thema 2: Effiziente Berechnung der transitiven Reduktion eines DAGs [A, P, E, G]

Zu jedem gerichteten, azyklischen Graphen (DAG) G kann die eindeutige transitive Reduktion G_{tr} ermittelt werden. Bei dieser handelt es sich um den Graphen mit der kleinstmöglichen Anzahl Kanten, bei dem die transitive Hülle von G_{tr} identisch zur transitiven Hülle von G ist. Umgangssprachlich beschreibt die transitive Reduktion eines Graphen den kleinsten (in Bezug auf die Kanten) Subgraphen, bei dem die Erreichbarkeit zwischen den Knoten unverändert bleibt. Ein beispielhafter einfacher Algorithmus beinhaltet das Durchführen einer Tiefensuche für jeden Knoten des Graphen, um Kanten zu bestimmen, die entfernt werden können. Die Einfachheit des Algorithmus wird durch vergleichsweise schlechte asymptotische Komplexität im Sinne der Laufzeit erkauft. Andere einfache Ansätze zeigen Schwächen bei der asymptotischen Komplexität im Sinne des Speicherplatzverbrauchs. Sie sind daher nicht für große Graphen geeignet.

Aufgabe des Praktikums ist es neue, bereits existierende, Algorithmen zur effizienteren Lösung des Problems zu implementieren. Dazu gehört beispielsweise [Tan+20], plus darin referenzierte Literatur. Bei der Implementierung soll die Korrektheit und Performanz ausführlich ausgewertet werden.

Programmiersprache: C++

Thema 3: Verbessertes Fahrrad- und Fußgänger-Routing

Das Open-Source Routing `osr`[4] soll um folgende Funktionalitäten erweitert werden:

[2] `https://github.com/felixguendling/cista`
[3] `https://github.com/google/benchmark`
[4] `https://github.com/motis-project/osr`

- Fahrrad- und Fußgängerpofil sollen jeweils das Höhenprofil berücksichtigen. Hierfür werden DEM (Digital Elevation Model) Daten aus Radar bzw. LiDAR Aufnahmen eingelesen und als Kanteninformation verfügbar gemacht (Steigung und Gefälle für jede Kante).
- Es werden verschiedene Fahrrad und Fußgängerprofile erstellt, die Steigungen und Gefälle jeweils bestrafen oder belohnen.
- Weitere in OpenStreetMap vorhandene Weg-Eigenschaften (wie z.B. Untergrund) werden in den Profilen bestraft bzw. belohnt.

Programmiersprache: C++

Literatur

[Tan+20] Xian Tang u. a. „One Edge at a Time: A Novel Approach Towards Efficient Transitive Reduction Computation on DAGs“. In: *IEEE Access* 8 (2020), S. 38010–38022. DOI: 10.1109/ACCESS.2020.2975650.