Universität Karlsruhe
Fakultät für Informatik
ITI Wagner

Algorithmen für Routenplanung
Sommersemester 2009
Daniel Delling, Thomas Pajor

Viertes Theorie- und Praxis-Übungsblatt

Ausgabe: 22. Juni 2009
Abgabe: 3. Juli, in der Vorlesung oder in Raum 322 (Informatik-Hauptgebäude, 3. Stock)

Theorieteil

Aufgabe 1: SHARC-Routing **

Gegeben sei ein Graph $G = (V, E, \text{len})$ und k-level multi-level Partition $\mathcal{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_k\}$. Betrachten Sie die Vorberechnung des SHARC-Algorithmus.

(a) **1. Schritt**: Kontraktion und Setzen von Arc-Flags.

Im aktuellen Iterationsschritt werde gerade eine Zelle C auf Level i verarbeitet (also $C \in \mathcal{P}_i$). Wie können bei Kanten $e \in C$, die durch Knotenreduktion entfernt werden, die Arc-Flags auf allen(!) Leveln korrekt gesetzt werden? Begründen Sie Ihre Antwort.

(b) **2. Schritt**: Verfeinerung der Arc-Flags.

Geben Sie ein effizientes Verfahren zur Verfeinerung der Arc-Flags an. Benutzen Sie dazu lokale Suchen.

Hinweis: Gehen Sie von einer 1-Level-Partition aus, und verallgemeinern Sie Ihren Ansatz auf k Level.

(c) **3. Schritt**: Arc-Flag-Kompression.

Zeigen Sie die Korrektheit der Arc-Flags-Kompression wie in der Vorlesung eingeführt. Beschränken Sie sich der Einfachheit halber bei Ihrem Beweis auf 1-Level-Partitionen.

Aufgabe 2: Transit-Node Routing & Arc-Flags ***

Wir wollen den Transit-Node-Algorithmus zielgerichtet machen. Es seien dazu zwei Level $\mathcal{T}_1 \subseteq \mathcal{T}_0$ (wobei $\mathcal{T}_0 = V$) von Transit-Nodes gegeben.

Wir bedienen uns folgender Intuition: Seien zu einer s-t-Anfrage $\overrightarrow{A}(s)$ und $\overleftarrow{A}(t)$ die Vorwärts- und Rückwärts-Access-Nodes. Knoten $a \in \overrightarrow{A}(s)$ sollen bei der Suche nicht betrachten werden, wenn es *keinen* kürzesten Weg von s gibt, der "in Richtung" Ziel t führt. Analog sollen für die Rückwärtssuche keine Knoten $a' \in \overleftarrow{A}(t)$ betrachtet werden, wenn es keinen kürzesten Weg von t nach s auf dem Rückwärtsgraphen $\overleftarrow{G}$ gibt.

Um dies zu ermöglichen, sollen für Transit-Knoden aus $\mathcal{T}_1$ jeweils ein Bitvektor der Länge k benutzt werden, wobei die Menge von Transit-Knoten in k Zellen partitioniert werden soll.

(a) Zu einer s-t-Anfrage (bei der der Locality-Filter nicht zuschlägt) werden durch deren Access-Nodes Mengen von Regionen induziert. Wann ist ein Table-Lookup zwischen zwei Access-Nodes $a_1 \in \overrightarrow{A}(s)$ und $a_2 \in \overleftarrow{A}(t)$ überflüssig? Welche Information soll daher an den Transit-Knoten gespeichert werden?

(b) Geben Sie ein Verfahren zur Berechnung der Bitvektoren an.

(c) Wie lässt sich die Transit-Node-Query modifzieren um von den vorberechneten Zusatzinformationen gebrauch zu machen?

Praxisteil

In dieser Übung soll die CHASE-Query implementiert werden. Laden Sie sich dazu das aktuelle Framework von der Vorlesungsseite herunter und erweitern Sie die Contraction-Hierarchies-Query in der Datei `CHASE_Dijkstra.h`. Weiterhin geben wir Ihnen für die Contraction-Hierarchy-Graphen aus dem Framework des letzten Blatts bereits vorberechnete Vorwärts- und Rückwärtsflags vor, die über `-f` respektive `-F` übergeben werden.

(a) Führen Sie die notwendigen Änderungen an der Contraction-Hierarchy-Query durch damit jeweils Kanten, bei denen die passenden Vorwärts- bzw. Rückwärtsflags nicht gesetzt sind, geprunt werden.

(b) Evaluieren Sie die Laufzeit sowohl gegenüber DIJKSTRA's Algorithmus alsauch gegenüber dem Contraction-Hierarchies-Algorithmus vom letzten Übungsblatt. Welchen zusätzlichen Speed-Up können Sie durch die Kombination mit Arc-Flags erreichen?

Schicken Sie Lösungsvorschläge per E-Mail an `pajor@ira.uka.de`. Dabei ist es ausreichend Ihre Version der Datei `CHASE_Dijkstra.h` mitzuschicken.