Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 11
21. Juli 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: In der jeweiligen Tutorübung

Hausaufgabe 1

Implementieren Sie in der Klasse `SSSP` den Algorithmus von Dijkstra zur Berechnung der kürzesten Wege von einem Knoten s zu allen anderen Knoten in dem ungerichteten Graphen G.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `SSSP` aber auf keinen Fall deren Interface `ISSSP`.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klassen den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_dij` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.
Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.
Hinweis: Verwenden Sie grundlegende Datenstrukturen, wie sie Java bereitstellt, und implementieren Sie diese *nicht* neu.

Hausaufgabe 2

Implementieren Sie in der Klasse `MST` den Algorithmus von Jarnik und Prim zur Berechnung eines Minimum Spanning Tree beginnend von einem Knoten s von einem ungerichteten Graphen G.
Der Algorithmus von Jarnik und Prim wählt zu einer Menge V' den Knoten der über eine Kante mit dem geringsten Gewicht erreichbar ist und fügt Ihn der Menge V' hinzu. Gestartet wird der Algorithmus mit $V' = s$.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `MST` aber auf keinen Fall deren Interface `IMST`.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klassen den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_mst` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.
Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.
Hinweis: Verwenden Sie grundlegende Datenstrukturen, wie sie Java bereitstellt, und implementieren Sie diese *nicht* neu.

Aufgabe 1

Argumentieren Sie, warum es unklug ist, bei der Berechnung eines kürzesten Weges von s nach t in einem Graphen mit echt positiven Gewichten alle möglichen (einfachen) Wege aufzuzählen und deren Distanz zu berechnen.
Benutzen Sie dazu exemplarisch ein Gitter der Größe n. Die n^2 Knoten werden durch die Tupel (x, y) mit $x, y \in [n]$ identifiziert.
Die Kanten $((x, y), (x + 1, y))$ seien gerichtet während die Kanten $((x, y), (x, y + 1))$ bidirektional verlaufen. Berechnen Sie die Anzahl aller Pfade von $s = (1, 1)$ zu $t = (n, n)$.

Lösungsvorschlag

Die Anzahl der Pfade in einem Gitter der Größe n ist exponentiell in n. Daher benötigt jeder Algorithmus, der alle Pfade auflistet also auch exponentiell viel Zeit. (Dies entspricht nicht der üblichen Vorstellung von effizienter Berechnung (polynomiell in der größe der Eingabe).
Die Anzahl der Pfade ergibt sich wie folgt:
In jeder Zeile, gibt es n Möglichkeiten von Zeile x in die Zeile $x + 1$ zu kommen (An der Positionen 1 bis n). Eine solche Auswahl kann in $n - 1$ Zeilen getroffen werden, da das Ziel in der letzten Zeile schon vorgegeben ist. So ergeben sich n^{n-1} verschiedene, einfache Wege vom Knoten $(1, 1)$ zu Knoten (n, n)

Aufgabe 2

Geben Sie einen möglichst effizienten Algorithmus an, der die *Anzahl* der kürzesten Wege von einem Knoten s zu allen anderen Knoten in einem ungerichteten Graphen G mit echt positiven Gewichten berechnet.

Lösungsvorschlag

Wir modifizieren den schon bekannten Dijkstra-Algorithmus. Für jeden Knoten merken wir uns in einem Array N die Anzahl der kürzesten Wege zu den jeweiligen Knoten. Zu dem Startknoten s existiert genau ein Weg.
Wird ein neuer kürzester Weg zu einem Knoten v von dem Knoten w aus entdeckt, so wird die Anzahl der kürzesten Wege $N[v]$ zu Knoten v mit der Anzahl der kürzesten Wege von w abgespeichert (`N[v]:=N[w]`). Wird der Knoten v erneut von einem anderen Knoten x mit der gleichen Distanz zu s „entdeckt“ so wird der werden die Anzahl der Wege, die über x zu v führen zu $N[v]$ dazugerechnet `N[v]+=N[x]`).