Universität Karlsruhe
Fakultät für Informatik
ITI Prof. Dr. D. Wagner

Praktikum
WS 2007/08
7. Dezember 2007

Übungsblatt 3

Abgabe: **Vortrag (19. Dezember 2007)**

Aufgabe 1: Großer Datensatz

Laden Sie den Straßengraphen der USA (512MB) herunter. Zu finden unter:
http://i11www.iti.uni-karlsruhe.de/~delling/tmp/usa.tar.bz2

Schreiben Sie zwei Programme, die jeweils die Laufzeit messen. Benutzen Sie hierzu `timer.h`.

(a) Messen Sie, wie lange ihre Datenstruktur zum Einlesen des USA-Graphen benötigt.

(b) Messen Sie, wie lange ihre Datenstruktur zum iterieren aller Kanten benötigt.

Was ist problematisch an dieser Zeitmessung?
Messen Sie jeweils den Speicherverbrauch ihrer Programme mit Hilfe von valgrind.
http://valgrind.org/

Aufgabe 2: Demandfiles

Um die Vergleichbarkeit der Datenstrukturen zu gewährleisten, benutzen wir sogenannte *demandfiles*. Diese files beinhalten pro Zeile eine Operation, die ausgeführt werden soll, wobei ein Wert der Überprüfung des Ergebnisses dient.
Für den statischen Fall gibt es 5 Arten von Operationen:

- `t [1|0]`: starte/beende Zeitmessung
- `e [s]`: lese Datei von s
- `B [u] [c]`: führe von Knoten u eine Breitensuche aus, die nur die Zusammenhangskomponente von u besucht. c entspricht hierbei der inversen closeness[1]
- `D [u] [s]`: führe von Knoten u eine Tiefensuche aus, die nur die Zusammenhangskomponente von u besucht. s entspricht der Anzahl besuchter Knoten.
- `d [u] [v] [d]`: bestimmte den kürzesten Weg von u nach v mit Hilfe eines einfachen *unidirektionalen* Dijkstra's. d gibt die Distanz zwischen u und v an.

Generell sollte klar sein, dass eine Operation *ohne* Wissen über spätere Operationen ausgeführt werden sollte. Dies ist vor allem für die dynamische Datenstruktur von großer Bedeutung.
Schreiben Sie ein Programm, dass ein demand-file einliest und dann die entsprechenden Operationen ausführt. Messen Sie die Zeit, die zur Abarbeitung des demand-files `example.demands` (zu finden im SVN unter data) von `t 1` bis `t 0` benötigt wird. Messen Sie den Speicherverbrauch mit Hilfe von valgrind.

BITTE WENDEN!

[1] inverse closeness eines Knoten u: $\sum_v d(u,v)$, wobei $d(u,v)$ der (ungewichtete) Abstand von v zu u bezüglich der Breitensuche ist.

Aufgabe 3: Implementationsskizze

Überlegen Sie sich eine effiziente Erweiterung ihrer statischen Datenstruktur, die folgende Operationen unterstützen soll:

- `insert_edge(u,v,w)`: füge Kante von u nach v mit Gewicht w ein.
- `delete_edge(u,v)`: lösche Kante von u nach v
- `update_weight(u,v,w)`: verändere das Gewicht der Kante (u, v) auf w.
- `delete_node(u)`: lösche Knoten u.
- `insert_node(u)`: füge Knoten u ein.

Welche Über- und Rückgabewerte sind nötig/sinnvoll?
Welche Sonderfälle müssen abgefangen werden? Was muss beachtet werden, wenn der Graph einfach bleiben soll? Was passiert, wenn ein Knoten gelöscht wird, der noch ein- und ausgehende Kanten besitzt? Sollen batch-updates unterstützt werden? Besprechen Sie weitere Ideen/Vorschläge/Probleme mit Ihrem Betreuer.