Programmierpraktikum Diskrete Optimierung (P2C1)

Thema: Kostenminimale Flüsse

Homepage der Veranstaltung:
`http://www.or.uni-bonn.de/lectures/ss09/proprakt_ss09.html`

Ansprechpartner: Christoph Bartoschek, E-Mail: `bartoschek@or.uni-bonn.de`

Allgemeine Bemerkungen:

In diesem Programmierpraktikum sollen verschiedene effiziente Algorithmen zur Berechnung von kostenminimalen Flüssen implementiert werden. Jeder Teilnehmer und jede Teilnehmerin soll (alleine oder in einer kleinen Gruppe) jeweils einen Flußalgorithmus implementieren. Dabei sind auch Datenstrukturen zur effizienten Abspeicherung von Graphen und (soweit notwendig) weitere Hilfsdatenstrukturen (z.B. Heaps) selbst zu implementieren. Es wird erwartet, daß effiziente Datenstrukturen verwendet werden und die Programme auch die jeweils erreichbare Laufzeit einhalten.
Die Programme in diesem Praktikum sind in C++ zu schreiben.
Eine Übersicht und Einführung in die Flußprobleme findet sich in [1] und [4].

Ablauf und Aufgabenstellungen:

Zunächst sollten Sie die Algorithmen, die Sie implementieren werden, durcharbeiten. In der Woche vom **9. März 2009 bis zum 13. März 2009** (die genauen Termine werden in der Vorbesprechung festgelegt) werden dann in Einzelbesprechungen die Details der Aufgaben definiert. In diesen Besprechungen sollen auch inhaltliche Fragen geklärt werden.
Es ist eine Einarbeitungsaufgabe zu bearbeiten, welche die Lösung der eigentlichen Aufgabe vorbereiten soll. **Die Lösung der Einarbeitungsaufgabe muß spätestens bis zum 20. April 2009 abgegeben werden.** Anschließend wird es eine Zwischenbesprechung geben, in der die Ergebnisse der Einarbeitungsaufgabe und Fragen zu den weiteren Aufgaben besprochen werden sollen. Wenn Fragen oder Probleme auftauchen, können Sie sich natürlich auch sonst jederzeit melden.

Zu implementierende Algorithmen:

1. **Netzwerksimplex:** Implementieren Sie den Netzwerksimplex-Algorithmus, wie er in [4] beschrieben ist (siehe auch [1]).
   Einarbeitungsaufgabe: Berechnen Sie eine initiale Baumlösung

2. **Minimum-Mean-Cycle-Cancelling-Algorithmus:** Implementieren Sie den Minimum-Mean-Cycle-Cancelling-Algorithmus, wie er in [4] beschrieben ist. Zur Berechnung der Kreise mit minimalem Durchschnittsgewicht verwenden Sie den in [6] beschriebenen Algorithmus.
   Einarbeitungsaufgabe: Implementieren sie den in [6] vorgestellten Algorithmus ohne Verwendung von Heaps.

3. **Orlins Algorithmus:** Implementieren Sie Orlins Algorithmus, wie er in [4] beschrieben ist.
   Einarbeitungsaufgabe: Implementieren sie den Sukzessive-Kürzeste-Wege-Algorithmus.
   Mögliche Zusatzaufgabe: Implementieren sie den Transportation-Flow-Algorithmus, wie er in [2] beschreiben ist.

4. **Cost-Scaling-Algorithmus:** Implementieren Sie den Cost-Scaling-Algorithmus, wie er in [3] beschrieben ist.
   Einarbeitungsaufgabe: Berechnen Sie einen zulässigen b-Fluß mit dem Push-Relabel-Algorithmus von Goldberg und Tarjan (siehe [4]).

5. **Forest-Cut-Algorithmus:** Implementieren Sie den Forest-Cut-Algorithmus, wie er in [5] beschrieben ist.
   Einarbeitungsaufgabe: Schreiben Sie ein Programm, das einen maximalen s-t-Fluß bestimmt.

Die Lösungen der Gesamtaufgaben sind spätestens bis zum 10. Juli 2009 abzugeben.

Datenformat:

Die Instanzen sind im DIMACS-Format gegeben:
Die Knoten sind von 1 bis n durchnumeriert, Kanten werden als geordnete Paare von Knotennummern angegeben.

Zeileneinträge:

`c <Text>` - Eine Kommentarzeile

`p min <nodes> <arcs>` - Eine Aufgabenzeile

`<nodes>` - Zahl der Knoten

`<arcs>` - Zahl der Kanten

`n <id> <flow>` - Eine Knotenzeile,

`<id>` - Knotenindex

`<flow>` - b-Wert des Knotens

`a <v> <w> <low> <cap> <cost>` - Eine Kantenzeile,

`<v>` - Index des Startknotens

`<w>` - Index des Endknoten

`<low>` - Untere Schranke für den Fluß auf der Kante

`<cap>` - Obere Schranke für den Fluß auf der Kante; die Kante hat unendliche Kapazität falls `<cap>` < 0.

`<cost>` - Kosten der Kante

In einer korrekten Eingabe steht genau eine Aufgabenzeile. Es kann keine Knotenzeile vor der Aufgabenzeile stehen und keine Kantenzeile vor eine Knotenzeile.
Alle Zahlen in den betrachteten Eingaben sind ganze Zahlen, die dem Betrage nach nicht größer als 2^{31} sind. Außerdem darf angenommen werden, daß alle unteren Schranken `<low>` für den Fluß auf einer Kante 0 sind.
Die Programme sollen so aufgerufen werden können, daß man ihnen den Namen einer Datei mit einer Instanz im oben angegeben Format übergibt.
In der Ausgabe soll in der ersten Zeile genau eine ganze Zahl stehen, die den Wert des berechneten Flusses angibt. Jeder weitere Zeile der Ausgabe spezifiziert den Fluß auf genau einer Kante. Dabei nehmen wir an, daß die Kanten in der Reihenfolge, in der sie in der Eingabedatei stehen von 1 bis m durchnumeriert sind. Eine solche Zeile hat also das Format

```
<row_index> <flow>
```

Kanten mit Fluß 0 sollen dabei nicht ausgegeben werden.

Literatur

[1] R. Ahuja, T. Magnanti, J. Orlin *Network Flows*, Prentice Hall, 1993

[2] U. Brenner *A faster polynomial algorithm for the unbalanced Hitchcock transportation problem.* Operations Research Letters, vol. 36, 4, 408–413, 2008.

[3] A. Goldberg *An efficient implementation of a scaling minimum-cost flow algorithm*, Journal of Algorithms, 22, 1–29, 1997

[4] B. Korte, J. Vygen, *Kombinatorische Optimierung*, Springer, 2008

[5] J. Vygen *On dual minimum cost flow algorithms*, Mathematical Methods of Operations Research, 56, 101–126, 2002

[6] N. Young, R. Tarjan, J. Orlin *Faster Parametric Shortest Path and Minimum Balance Algorithms* Networks 21(2), 205–221, 1991