Wintersemester 2010/11
Prof. Dr. J. Vygen
Dr. U. Brenner

Forschungsinstitut für Diskrete Mathematik
Universität Bonn

Einführung in die Diskrete Mathematik

3. Übung

1. Sei G ein ungerichteter Graph mit Kantengewichten $c : E(G) \to \mathbb{R}$.
   Wie lassen sich die folgenden Probleme möglichst effizient lösen?

   (a) Sei $v \in V(G)$ ein Knoten. Gesucht ist ein aufspannender Baum, in dem v kein Blatt ist und der unter allen aufspannenden Bäumen, in denen v kein Blatt ist, minimales Gewicht hat.

   (b) Man bestimme die Menge aller Kanten $e \in E(G)$, für die es einen aufspannenden Baum T_e mit minimalem Gewicht gibt, so dass e in T_e enthalten ist.

   (c) Man bestimme einen aufspannenden zusammenhängenden Teilgraphen von G mit minimalem Gewicht.

   (d) Man bestimme einen aufspannenden Baum T für G, dessen maximales Kantengewicht minimal ist. (4 Punkte)

2. Sei G ein gerichteter Graph mit Kantengewichten $c : E(G) \to \mathbb{R}_+$, und sei $v \in V(G)$, so dass alle Knoten in G von v aus erreichbar sind. Es bezeichne A die Menge aller gewichtsminimalen aufspannenden Arboreszenzen mit Wurzel v, und B die Menge aller Kürzeste-Wege-Arboreszenzen mit Wurzel v (d.h. für jedes $T \in B$ und jeden Knoten $w \in V$ ist der v-w-Weg in T ein kürzester v-w-Weg in G). Gilt stets $A \subseteq B$? Gilt stets $B \subseteq A$? (4 Punkte)

3. Implementieren Sie einen Algorithmus (basierend auf dem Moore-Bellman-Ford-Algorithmus), der zu einem gegebenen gerichteten Graphen G und $c : E(G) \to \mathbb{R}$ entweder ein zulässiges Potential oder einen negativen Kreis berechnet. Ihr Programm soll Laufzeit $O(nm)$ haben. Implementieren Sie Ihr Programm so, dass in jeder Iteration nur noch die Kanten (v, w) betrachtet werden, für die $l(v)$ in der vorigen Iteration verändert wurde. (12 Punkte)

Abgabe:
Aufgaben 1 und 2: Dienstag, den 2.11.2010, **vor** der Vorlesung.
Aufgabe 3: Dienstag, 9.11.2010, **vor** der Vorlesung.

Hinweise zur Programmierübung:

Einlesen der Daten: Dem Programm muss beim Aufruf der Name einer Datei übergeben werden. Ein Aufruf hat also die Form

```
<programmname> <dateiname>
```

Eine gültige Datei, die eine Instanz beschreibt, hat das folgende Format:

```
Knotenanzahl
Kantenzahl
Knoten0a Knoten0b Gewicht0
Knoten1a Knoten1b Gewicht1
...
```

Die Einträge der Datei sind ausschließlich ganze Zahlen. Sie können voraussetzen, dass die Summe der Absolutbeträge aller Zahlen in der Eingabe kleiner als 2^{31} ist. In den ersten beiden Zeilen steht jeweils eine einzelne natürliche Zahl (größer als 0), welche in der ersten Zeile die Anzahl der Knoten und in der zweiten die Anzahl der Kanten angibt. Wir nehmen an, dass, wenn wir n Knoten haben, die Knoten von 0 bis $n-1$ durchnumeriert sind. Jede weitere Zeile spezifiziert genau eine Kante. Die ersten beiden Einträge einer Zeile sind zwei verschiedene nichtnegative ganze Zahlen, welche die Nummern der Endknoten der Kante sind. Der dritte Eintrag in der Zeile ist eine ganze Zahl, die das Gewicht der Kante bezeichnet. Es können parallele Kanten vorkommen, und der Graph muss nicht zusammenhängend sein.

Ausgabeformat: Die erste Zeile der Ausgabe muss genau eine Zahl enthalten, nämlich 0, wenn ein negativer Kreis gefunden worden ist, und 1 sonst.

Wenn ein negative Kreis gefunden wurde, soll der Rest der Ausgabe einen solchen Kreis kodieren. In jeder Zeile soll dabei der Index eines Knoten eingetragen werden, so dass aufeinanderfolgende Knoten durch eine Kante des Kreises verbunden sind.

Wenn kein negativer Kreis gefunden wurde, sollen die weitere Zeilen der Ausgabe $\pi(0), \ldots, \pi(n-1)$ enthalten, so dass π ein zulässiges Potential ist.

Beispiel 1: Eine Eingabedatei für einen Graphen mit 4 Knoten und 5 Kanten kann so aussehen:

```
4
5
0 1 2
1 3 4
0 3 7
2 0 -1
2 3 2
```

Die Ausgabe der Programms kann dann so aussehen:

```
1
-1
0
0
0
```

Beispiel 2: Eine Eingabedatei für einen Graphen mit 4 Knoten und 4 Kanten kann so aussehen:

```
4
4
0 3 -3
1 0 -1
3 1  2
1 2  1
```

Die Ausgabe der Programms kann dann so aussehen:

```
0
3
1
0
```

Das Programm muss in C oder C++ geschrieben sein. Es muss korrekt arbeiten und ohne Fehlermeldung kompiliert werden können. Der Code muss auf einem gängigen Linuxsystem funktionieren. Algorithmen aus externen Bibliotheken dürfen nicht verwendet werden.

Abgabe: Der Quelltext der Programms muss bis 9. November, 16:15 Uhr per E-Mail beim jeweiligen Tutor eingegangen sein. Außerdem ist bis zu diesem Zeitpunkt ein Ausdruck des Quelltextes zusammen mit den Theorieaufgaben abzugeben.

Testinstanzen befinden sich auf der Seite

`http://www.or.uni-bonn.de/lectures/ws10/edm_10_uebung.html`