Technische Universität München
Institut für Informatik
Prof. Dr. Ernst W. Mayr

SS 1998
Aufgabenblatt 3
25. Mai 1998

Praktikum Algorithmen-Entwurf

(Abgabetermin: Montag, den 8. Juni 1998, 24.⁰⁰ Uhr)

Minimale Spannbäume II

Aufgabe 1 Algorithmus von Kruskal ohne LEDA

Implementieren Sie ohne Verwendung von LEDA den Algorithmus von Kruskal zur Berechnung minimaler Spannbäume in zusammenhängenden, ungerichteten Graphen mit ganzzahligen Kantengewichten. Ihr Programm soll den gewichteten Graphen aus einer Eingabedatei lesen, deren Name als Kommandozeilenargument übergeben wird, und das Gewicht eines minimalen Spannbaums auf den Standardausgabe-Stream ausgeben. Das Format der Eingabedatei ist auf der Rückseite dieses Aufgabenblattes erklärt.

Aufgabe 2 Minimale Spannbäume mit Nebenbedingung

Gegeben sei wieder ein zusammenhängender, ungerichteter Graph mit ganzzahligen Kantengewichten. Entwickeln Sie einen Algorithmus, der *für jede Kante e des Graphen* das Gewicht eines Spannbaums berechnet, der die Kante e enthält und unter allen Spannbäumen, die e enthalten, minimales Gewicht hat.
Ziel ist natürlich ein möglichst effizienter Algorithmus, und die Bewertung erfolgt (zum Teil) anhand von Laufzeittests mit sehr großen Eingaben. Tip: Es könnte nützlich sein, die Lösung von Aufgabe 1 als Unterroutine zu verwenden, möglichst aber ohne sie m-mal aufzurufen.
Der Algorithmus ist in C oder C++ ohne Verwendung von Funktionsbibliotheken wie LEDA zu implementieren (nur Standard-Bibliotheken wie `iostream`, `stdlib`, `stdio`, etc. sind zugelassen). Die Eingabe ist wie bei Aufgabe 1 aus einer Datei zu lesen, deren Name als Kommandozeilenargument angegeben wird. Da der Algorithmus m Baumgewichte berechnet, besteht die Ausgabe aus m Zahlen, die dezimal über den Standardausgabe-Stream auszugeben sind, und zwar genau in der Reihenfolge der Kantenliste im Eingabefile, je gefolgt von einem newline Symbol (`\n`). Bitte beachten Sie, daß die einzelnen Zeilen der Ausgabe jeweils nur eine Zahl enthalten dürfen, keine weiteren Symbole.

Hinweis:

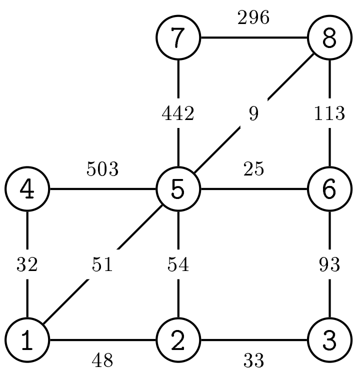
Es braucht nur die Lösung zu Aufgabe 2 abgegeben zu werden. Bitte fügen Sie Ihrer Lösung auch eine *kurze* Beschreibung des Algorithmus bei. Ein detaillierter und formaler Korrektheitsbeweis wird nicht verlangt. Sollten Sie graphentheoretische Lemmata ohne Beweis verwenden, so geben Sie diese bitte an. Achten Sie auch auf konstante Faktoren in der Laufzeit Ihres Programms. Exzessiver Speicherverbrauch kann zu Plattenzugriffen führen und die Laufzeit wesentlich erhöhen.

Eingabeformat:

Die Eingabe ist eine Textdatei, die aus $n+m+1$ Zeilen besteht, wobei $n = |V|$ und $m = |E|$. Die erste Zeile enthält die beiden Zahlen n und m, durch ein Leerzeichen getrennt. Die folgenden n Zeilen entsprechen den von 1 bis n numerierten Knoten und enthalten jeweils eine Zahl, nämlich den Offset (Index, zwischen 1 und $m+1$) in die Liste der Kanten. Der Index zeigt in der Kantenliste auf die erste Kante, die zum gegebenen Knoten adjazent ist. Die folgenden m Zeilen enthalten die Kantenliste. Jede Zeile entspricht einer Kante und besteht aus zwei Zahlen, von denen die erste Zahl den zweiten Endknoten der Kante angibt (also eine Zahl zwischen 1 und n ist) und die zweite Zahl das Kantengewicht (eine positive, ganze Zahl). Der erste Endknoten der Kante ist durch ihre Position in der Liste und die Offsets festgelegt. Jede ungerichtete Kante taucht nur einmal in der Kantenliste auf, d.h. nur bei einem der beiden Endknoten.

Beispiel:

`8 12`	
`1`	
`4`	
`6`	
`7`	
`8`	
`11`	
`12`	
`13`	
`2 48`	an Knoten 1
`4 32`	
`5 51`	
`3 33`	an Knoten 2
`5 54`	
`6 93`	an Knoten 3
`5 503`	an Knoten 4
`6 25`	an Knoten 5
`7 442`	
`8 9`	
`8 113`	an Knoten 6
`8 296`	an Knoten 7



Dieser Graph liegt als Eingabedatei `tree1.txt` im üblichen Verzeichnis auf den hpmayr-Rechnern bereit. Weitere Eingabedateien und zum Teil auch die zugehörigen Lösungen sind im selben Verzeichnis und im WWW verfügbar (Dateinamen `tree*.txt` oder `tree*.sol`).