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

WS 1999/2000
Aufgabenblatt 2
29.10.2001

Praktikum Algorithmen-Entwurf

(Abgabetermin: Montag, den 5.11.2001, $12.^{00}$ Uhr)

Minimale Spannbäume

Aufgabe 1 Algorithmen von Kruskal und Prim (mit Animation)

Implementieren und animieren Sie den Algorithmus von Kruskal und den Algorithmus von Prim zur Berechnung eines minimalen Spannbaums in einem ungerichteten, zusammenhängenden Graphen G mit positiven, ganzzahligen Kantengewichten. Es soll am Bildschirm sichtbar sein, in welcher Reihenfolge die Kanten von G bearbeitet werden und welche Kanten bereits in den Spannbaum eingefügt wurden. Für den Algorithmus von Kruskal steht Ihnen frei, entweder die Implementierung mittels Priority-Queue oder die Implementierung mit Sortierung der Kanten zu wählen.

Als Eingabe stehen die Graphen `mst1.gw` bis `mst4.gw` an den üblichen Orten zur Verfügung. Die Kantengewichte dieser Graphen sind ganze Zahlen, die im User-Label in String-Form mit abgespeichert sind. Um diese Werte zwecks bequemer Weiterverarbeitung in einem `edge_array` abzulegen, kann eine Schleife der folgenden Form verwendet werden:

```
#include <LEDA/stream.h>
...
edge_array<int> c(g);
forall_edges(e,g) {
    string s = gw.get_user_label(e);
    string_istream I(s);
    I>>c[e];
}
```

O D E R

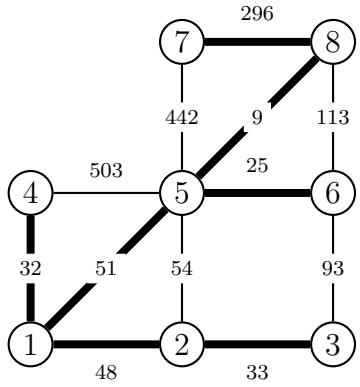
Aufgabe 2 Challenge-Aufgabe (ohne Animation)

Gegeben sei ein zusammenhängender, ungerichteter Graph mit positiven, ganzzahligen Kantengewichten. Entwickeln Sie einen möglichst effizienten Algorithmus, der für jede Kante e des Graphen das Gewicht w_e eines Spannbaums berechnet, der die Kante e enthält und unter allen Spannbäumen, die e enthalten, minimales Gewicht hat. Sie brauchen den Algorithmus nicht zu animieren. Ihr Algorithmus darf beliebige Klassen und Funktionen aus LEDA verwenden, also z.B. auch die Funktion MIN_SPANNING_TREE zur Berechnung eines minimalen Spannbaums. Bitte fügen Sie Ihrer Lösung auch eine *kurze* Beschreibung des Algorithmus bei.

— bitte wenden —

Aufgabe 2 (Fortsetzung)

Beispiel:



Minimaler Spannbaum: Gewicht 494

Kante	Spannbaum
48	494
32	494
51	494
33	494
54	497
93	536
503	946
25	494
442	640
9	494
113	582
296	494

Das Programm erhält als Aufrufparameter den Namen einer Datei, in der ein gerichteter Graph (der ungerichtet interpretiert werden soll!) mit ganzzahligen Kantengewichten gespeichert ist. Zum Einlesen des Graphen können Sie folgendes Programmstück verwenden:

```
main(int ac, char **av) {
    GRAPH<int,int> g;
    if (ac!=2) exit(1);
    g.read(av[1]);
    ...
    // Gewicht von Kante e kann als g[e] referenziert werden;
    // g.edge_data() liefert Referenz auf das edge_array<int>,
    // das die Kantengewichte enthält
}
```

Die Ausgabe Ihres Programms besteht bei einem Eingabegraphen mit m Kanten aus genau m Zahlen, nämlich den Werten w_e für jede Kante e. Ihr Programm soll diese Zahlen nacheinander auf dem `cout`-Kanal ausgeben, und zwar in der Reihenfolge, in der die Kanten von einer `forall_edges`-Schleife durchlaufen werden. Jede Zahl soll als String gefolgt von einem newline-Symbol ('\n') ausgegeben werden.

Ein Rahmenprogramm zum Einlesen der Eingabe, drei kleine Eingabegraphen mit zugehörigen Lösungen sowie ein Programm zum Erzeugen beliebig großer Eingaben wurden auf der Praktikums-Webseite

http://wwwmayr.in.tum.de/lehre/2001WS/algoprak/data.html

bereitgestellt.
Das Generierungsprogramm liegt auch als

/usr/wiss/raab/bin/gen

ausführbar auf den sunhalle-Rechnern für Sie bereit. Sie können zum Beispiel mit dem Befehl **/usr/wiss/raab/bin/gen 1000 mst1000.g** einen Graphen mst1000.g mit 1000 Knoten erzeugen.

Hinweis

Sie brauchen nur eine der beiden Aufgaben zu bearbeiten! Bei Aufgabe 2 werden Lösungen nur akzeptiert, wenn sie tatsächlich effizient sind. Dies bedeutet unter anderem, daß Ihre Lösung auch Eingaben mit 1000 Knoten noch in vernünftiger Zeit (d.h. auf den sunhalle-Rechnern deutlich unter einer Minute) verarbeiten können muß!