Uni Osnabrück / Theoretische Informatik — Prof. Dr. M. Chimani

Algorithm Engineering — Übungsprojekt
Aufgabenstellung & Übersicht

Wintersemester 2015/16

Aufgabenstellung: Gradbeschränkte Spannbäume

Gegeben ein einfacher Graph $G = (V, E)$ (keine Mehrfachkanten oder Schleifen) mit Kantengewichten $w\colon E \to \mathbb{N}_{\geq 1}$ und Maximalgraden $g\colon V \to \mathbb{N}_{\geq 1}$. Ein *gradbeschränkter Spannbaum* $T = (V, F)$, $F \subseteq E$, ist ein alle Knoten enthaltender Baum (zusammenhängender, kreisfreier Teilgraph von G), bei dem der Knotengrad jedes Knoten $v \in V$ maximal $g(v)$ ist, d.h.,

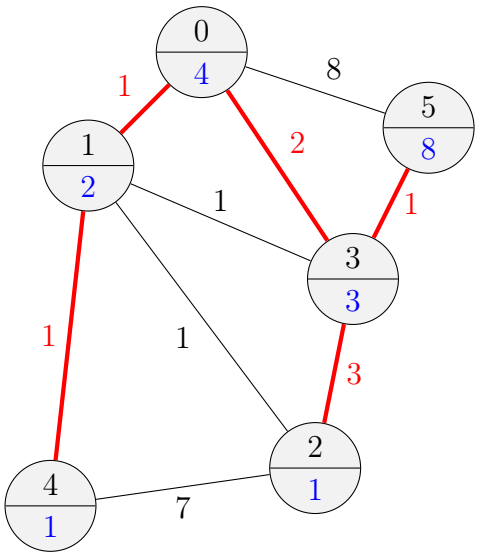
$$|\{e \in F \mid e \text{ ist inzident zu } v\}| \leq g(v) \qquad \forall v \in V.$$

Wir suchen einen gradbeschränkten Spannbaum mit minimalem Gewicht $w(T) := \sum_{e \in T} w(e)$.

Ein- und Ausgabeformat. Die Eingabedatei ist eine Textdatei. Die erste Zeile enthält den Namen bzw. eine kurze Beschreibung der Instanz. Die zweite bzw. dritte Zeile besteht aus einer positiven ganzen Zahl, die die Anzahl der Knoten (n) bzw. der Kanten (m) von G angibt. Die nächsten n Zeilen geben die Werte von g für die Knoten $V = \{0, 1, 2, \ldots, n-1\}$ (in dieser Reihenfolge) an. Die nächsten m Zeilen beschreiben jeweils eine Kante und enthalten immer 3 oder immer 4 Werte: die ersten drei Werte sind der Start- und Endknoten der Kante, sowie das Kantengewicht w. Der vierte Wert (falls er existiert) ist entweder 0 oder 1: Die Kanten die mit 1 markiert sind stellen eine zulässige Lösung (also einen gradbeschränkten Spannbaum) dar.

Beispiel einer Ein-/Ausgabedatei.
Die roten, kursiv gedruckten Werte („Lösung“) werden beim Einlesen der Datei i.A. ignoriert.

```
Meine Testinstanz
6
9
4
2
1
3
1
8
0 1 1 1
0 3 2 1
0 5 8 0
1 2 1 0
1 3 1 0
1 4 1 1
2 3 3 1
2 4 7 0
3 5 1 1
```



Programmiersprache. Der Code wird in C++ geschrieben, und muss mittels eines Makefiles (nur durch Eingabe von „`make`“) unter gcc 4.8 kompilieren. Um einen einfachen Austausch und Vergleich zu gestatten sind nicht-standard Zusatzbibliotheken zu vermeiden. Falls Sie meinen, unbedingt Bibliotheken zu benötigen, sprechen Sie vorher mit mir.

Ablaufplan

Phase 1. Benchmark-Instanzen & Framework **Deadline: ca. Mitte November**

(A) Sammeln bzw. generieren Sie Testinstanzen unterschiedlicher Größe, die in der Realität interessant sind/sein könnten. Versuchen Sie dabei i.A. sicherzustellen, dass die Lösungen teurer als der „normale“ Spannbaum ist, jedoch zulässige Lösungen gefunden werden können. (Dies sollte mehr Zeit und Aufmerksamkeit in Anspruch nehmen als (B)!)

und

(B) Erstellen Sie einen funktionierenden Prototypen ihres Programms. Hier werden die Spezifikationen nun (im Gegensatz zu den meisten anderen Spezifikationen des Projekts) sehr genau angegeben, damit wir in weiterer Folge die Algorithmen der verschiedenen Gruppen leicht untereinander austauschen werden können.

Sei `progname` ein beliebiger Name für ihr Programm:

- Der Aufruf „`progname`“ (ohne Parameter) gibt auf der Konsole eine Meldung über die möglichen Aufrufparameter aus.
- Der Aufruf „`progname -in dat1 -out dat2`“ liest die Eingabedatei `dat1` ein, führt den Algorithmus aus, und schreibt die Lösung in die Datei `dat2`. Auf der Konsole gibt der Algorithmus zwei Zahlen—durch Tabulator (`'\t'`) getrennt—aus: das Gewicht des gefundenen gradbeschränkten Teilbaums (bzw. `-1` falls keine zulässige Lösung gefunden wurde) und die benötigte Laufzeit.
- Der Aufruf „`progname -eval dat1`“ liest die Datei `dat1` ein. Falls die darin beschriebene Lösung zulässig ist, wird das Gewicht des gradbeschränkten Teilbaums auf die Konsole geschrieben; sonst der Text „`ERROR: `“, gefolgt von einer (hilfreichen) Fehlermeldung, warum die Lösung nicht korrekt ist.

Der Algorithmus muss zu diesem Zeitpunkt noch nicht „clever“ sein; eine beliebige zulässige Lösung zu generieren reicht aus (bei einigen Instanzen kann sogar das schon fehlschlagen). Anmerkung zur Laufzeit: Messen Sie nur die Zeit zwischen nach-dem-Laden und vor-dem-Schreiben!

Zur Deadline: Sammeln der Benchmark-Instanzen aller Gruppen

Phase 2. Mindestens zwei Heuristiken **Deadline: ca. Mitte Dezember**

Erstellen Sie mindestens zwei grundverschiedene Algorithmen, die das Problem heuristisch möglichst gut lösen. Wir betrachten an dieser Stelle nur „traditionelle“ Programmierungen, d.h. ohne Ausnutzen von Parallelität (mehrere Threads, o.ä.). Ihr Programm darf beliebige weitere Parameter zur Algorithmenkonfiguration akzeptieren (sollte aber auch mit Defaultwerten, ohne Extra-Parameter, funktioneren).

Zur Deadline: Vorstellung der Resultate inkl. Ausblick was sich verbessern ließe; Austausch der Algorithmen

Phase 3. Exakte Verfahren **Deadline: ca. Mitte Januar**

Details folgen.

Zur Deadline: Vorstellung der Resultate und Vergleiche ggü. den Heuristiken.

Phase 4. Verbesserte Algorithmen **Deadline: ca. letzte Vorlesungswoche**

Details folgen.

Zur Deadline: Vorstellung der Resultate