RWTH Aachen **SS 2016**
Lehrgebiet Theoretische Informatik Lösungsvorschlag 13
Rossmanith–Hensel–Kuinke–Sánchez 18. Juli 2016

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T41

2¢-Münzen:

Nehmen wir an, v sei mindestens 2. Da der Betrag durch 1¢-Münzen ausgedrückt wird (nach Annahme wird keine 2¢-Münze benutzt), sind mindestens zwei 1¢-Münzen vorhanden, die wir durch eine 2¢-Münze ersetzen können.

5¢-Münzen:

Nehmen wir nun an, v sei mindestens 5. Werden keine 5¢-Münzen benutzt, so läßt sich der Betrag darstellen als $v = 2c_2 + c_1$, wobei c_2 die Anzahl der 2¢- und c_1 die Anzahl der 1¢-Münzen bezeichnet. Die Anzahl c_1 kann nur die Werte 0 und 1 annehmen, denn ansonsten ist leicht eine kleiner Lösung mit einer weiteren 2¢-Münze gefunden (vgl. die zuvor gezeigte Optimaliztät für 2¢-Münzen). Betrachten wir diese beiden Fälle also separat:

- $c_1 = 0$: Aus $v \geq 5$ folgt, daß $c_2 \geq 3$ ist. Dann aber erhalten wir eine bessere Lösung, wenn wir drei 2¢-Münzen durch eine 5¢- und eine 1¢-Münze ersetzen.
- $c_1 = 1$: Aus $v \geq 5$ folgt, daß $c_2 \geq 2$ ist. Wir erhalten eine bessere Lösung, indem wir zwei 2¢-Münzen und die 1¢-Münze eine 5¢-Münze ersetzen.

Aufgabe T42

Wir wollen das Problem mittels dynamischer Programmierung lösen. Sei der Baum T und die Wurzel r. Wir bezeichnen als $T(v)$ den Teilbaum, der v als Wurzel hat. $T(r)$ ist demnach der gesamte Baum. Sei $A(v)$ die optimale Lösung in $T(v)$ und $B(v)$ die optimale Lösung in $T(v) - \{v\}$ (die optimale Lösung wenn v nicht Teil der Lösung ist). Die Lösung des Problems ist der Wert $A(r)$, den wir berechnen wollen.

Wir beginnen bei den Blättern l: $A(l) = w(l)$ und $B(l) = 0$. Sei im folgenden v ein Knoten mit m Kindern $u_1, \ldots, u_m$.

Die optimale Lösung in $T(v)$ ist nun die bessere der folgenden beiden Möglichkeiten:

- v nicht hinzuzufügen.
- v hinzuzufügen.

Falls wir v nicht wählen (Wert $B(v)$), können wir einfach die optimalen Lösungen der Teilbäume der Kinder zusammenfügen, d.h. $B(v) = A(u_1) + \cdots + A(u_m)$.

Im zweiten Fall darf man keines der Kinder von v hinzufügen, da sie eine Kante zu v besitzen. Deshalb müssen wir die B Werte der Kinder wählen, und dazu das Gewicht von v addieren. Die bessere der beiden Möglichkeiten finden wir mit der maximums funktion, d.h. $A(v) = \max\{B(v), w(v) + B(u_1) + \cdots + B(u_m)\}$.

Da jeder Knoten nur einmal durchlaufen wird und der Zeitaufwand in jedem Knoten Konstant ist beträgt die Laufzeit $O(n)$. Der Speicheraufwand ist ebenfalls $O(n)$ da für jeden Knoten nur die zwei Werte $A(v)$ und $B(v)$ gespeichert werden müssen.

Aufgabe H37

Wir verwenden einen Greedyalgorithmus, welcher, solange noch nicht betrachtete Kanten existieren, diese entweder zur Menge F hinzufügt oder verwirft. Dieses geschieht in absteigender Reihenfolge nach Gewicht der Kanten. Eine Kante wird F hinzugefügt, falls F keine zwei Komponenten verbindet, die bereits beide Kreise enthalten,und auch keinen zweiten Kreis in einer Komponente erzeugt; sonst wird diese Kante verworfen.

Algorithmus in Pseudo-Code:

1. Sortiere Kanten absteigend nach Gewicht, nenne sie $e_1, \ldots, e_m$.
2. Setze $F := \emptyset$ und $\mathit{kreis}[v] := 0$, $\mathit{comp}[v] := v$ für alle $v \in V$.
3. Für $1 \leq i \leq m$: Sei $e_i = \{u_i, v_i\}$.
   (a) Falls $\mathit{comp}[u_i] = \mathit{comp}[v_i]$ und $\mathit{kreis}[\mathit{comp}[u_i]] = 0$ (gleiche Komponente, noch kein Kreis vorhanden), setze $F := F \cup \{e_i\}$ und $\mathit{kreis}[\mathit{comp}[u_i]] := 1$.
   (b) Falls $\mathit{comp}[u_i] \neq \mathit{comp}[v_i]$ und $\mathit{kreis}[\mathit{comp}[u_i]] \wedge \mathit{kreis}[\mathit{comp}[v_i]] = 0$ (verschiedene Komponenten, wenigstens eine bisher ohne Kreis), setze $F := F \cup \{e_i\}$, $\mathit{comp}[v_i] := \mathit{comp}[u_i]$ sowie

$$\mathit{kreis}[\mathit{comp}[u_i]] := \mathit{kreis}[\mathit{comp}[u_i]] \vee \mathit{kreis}[\mathit{comp}[v_i]] = 0.$$

Die Korrektheit dieses Algorithmus folgt unmittelbar aus der in der Vorlesung bewiesenen Lemmata zu den auf gewichteten Matroiden basierenden Greedy-Algorithmen. Das Sortieren der Kanten benötigt Zeit $O(m \log m) = O(m \log n)$, da $m \leq n^2$ gilt. Die Schleife wird m mal ausgeführt. Die Komponenten eines Knotens können mittels einer Union-Find-Datenstruktur effizient in Zeit $O(\log n)$ ermittelt und aktualisiert werden, so daß die Gesamtlaufzeit tatsächlich $O(m \log n)$ beträgt.

Für beide Graphen ist die optimale Lösung eine Menge die alle Kanten enthält, außer eine die Gewicht eins hat.