Johannes Blömer
u.v.a.m.

Paderborn, den 2. August 2004

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 13

AUFGABE 1:
Sei $T_0 = E, T_1, \ldots, T_k$ die Kantenmengen, die der Algorithmus DELETE-MST(G, w) in Zeile 5 sukzessive erzeugt. Weiter setzen wir $G_i = (V, T_i), i = 0, \ldots, k$. Der Graph G_k muss ein Baum und damit ein Spannbaum von G sein. Wir zeigen nun durch vollständige Induktion, dass für alle $i = 0, \ldots, k$ der Graph G_i einen minimalen Spannbaum von $G = G_0$ enthält.
Für $i = 0$, enthält $G_0 = G$ sicherlich einen minimalen Spannbaum von G. Nun zum Induktionsschritt. Wir setzen also voraus, dass G_{i-1} einen minimalen Spannbaum von G enthält und wollen dann zeigen, dass dieses auch noch für G_i gilt. Da G_{i-1} einen minimalen Spannbaum von G enthält, ist jeder minimale Spannbaum von G_{i-1} auch noch minimaler Spannbaum von G. Sei nun $T_i = T_{i-1} - \{e\}$, wobei e eine schwerste Kante in T_{i-1} ist, die noch in einem Kreis K von G_{i-1} enthalten ist. Angenommen, e ist eine Kante eines minimalen Spannbaums B von G_{i-1}. Wir konstruieren dann einen minimalen Spannbaum, der e nicht enthält. Dieses zeigt dann, das auch noch G_i einen minimalen Spannbaum von G enthält. Entfernen wir e aus B, zerfällt B in zwei Zusammenhangskomponenten. Neben e muss nun der Kreis K noch eine zweite Kante e' enthalten, die die beiden Zusammenhangskomponenten verbindet. Es gilt $w(e') \leq w(e)$. Dann aber ist $B' = B - \{e\} \cup \{e'\}$ ebenfalls ein Spannbaum von G_{i-1}. Das Gewicht von B' ist nicht größer als das Gewicht von B. Damit ist B' ein minimaler Spannbaum von G_{i-1} und von G. Wie oben bereits argumentiet, folgt daraus die Korrektheit von DELETE-MST.

AUFGABE 2:
Wir speichern die Anzahl der Münzen, die beim Bezahlen benutzt werden, in einem Array $A[0..k]$. Dabei ist $A[j]$ die Anzahl der Münzen mit Nennwert c^j. Hier zunächst der Algorithmus GIERIGES-BEZAHLEN(b) in Pseudocode.

GIERIGES-BEZAHLEN(b)

```
1 for i ← k to 0
2     do A[i] ← ⌊b/c^i⌋
3        b ← b − c^i · ⌊b/c^i⌋
4 return A
```

Zunächst zur Laufzeit von Algorithnmus GIERIGES-BEZAHLEN. Pro Durchlauf der FOR-Schleife wird konstante Zeit benötigt. Da die Schleife k-mal durchlaufen wird, ist die Laufzeit insgesamt $\mathcal{O}(k)$.
Jetzt zeigen wir, dass GIERIGES-BEZAHLEN ein zulässige Lösung liefert, also

$$\sum_{i=0}^{k} A[i]c^i = b$$

gilt. Aus der Setzung von $A[i]$ in Zeile 2 folgt, dass zu jedem Zeitpunkt $b \geq 0$ gilt. Nun ist aber $c^0 = 1$. Damit muss nach Durchlauf der FOR-Schleife für $i = 0$ der Wert von b auf 0 gesetzt sein und GIERIGES-BEZAHLEN berechnet eine zulässige Lösung.
Schliesslich zeigen wir, dass die von GIERIGES-BEZAHLEN berechnete Lösung $A[0], \ldots, A[k]$ optimal ist. Sei $(d_0, \ldots, d_k) \in \mathbb{N}^k$ mit $\sum_{i=0}^{k} d_i c^i = b$ eine zulässige Lösung mit $d_i \neq A[i]$ für mindestens einen Index i. Sei dann l der größte Index mit $d_l \neq A[l]$. Dann muss gelten $d_l < A[l]$, denn andernfalls können die d_i keine zulässige Lösung sein. Weiter muss dann gelten $\sum_{i=0}^{l-1} d_i c^i \geq c^l$. Nun gilt aber $\sum_{i=0}^{l-1}(c-1)c^i < c^l$. Daher folgt, dass für mindestens ein j zwischen 1 und $l-1$ gilt $d_j \geq c$. Dann können wir aber die Lösung $(d_0, \ldots, d_k)$ verbessern, indem wir d_{j+1} um 1 erhöhen und d_j um c erniedrigen. Damit ist $(d_0, \ldots, d_k)$ keine optimale Lösung.

AUFGABE 3:
Die Grundidee des gierigen Algorithmus GREEDY-SCHEDULING ist einfach.

GREEDY-SCHEDULING

1. Wir sortieren die Jobs nach absteigender Belohnung. Zur Vereinfachung nehmen wir an, dass diese Sortierung durch $j_1, \ldots, j_n$ gegeben ist. Nun versuchen wir den Jobs in dieser Reihenfolge Startzeiten zuzuordnen. Dabei wird einem Job entweder eine Startzeit vor seiner Deadline zugewiesen oder er wird markiert. Markiert heisst dabei, dass der Algorithmus nicht mehr versuchen wird, diesen Job vor seiner Deadline zu starten.

2. Sind den ersten $i-1$ Jobs bereits Startzeiten zugewiesen worden oder sind sie markiert worden, so wird für den i-ten Job j_i der größte Zeitpunkt $0 \leq t_i < d_i$ bestimmt, an dem keiner der ersten $i-1$ Jobs bereits gestartet werden soll. Existiert ein solcher Zeitpunkt nicht mehr, so wird Job j_i markiert.

3. Sind alle n Jobs in 2. betrachtet worden, ist die Reihenfolge der Jobs, denen Startzeiten zugewiesen wurden, gegeben durch die aufsteigende Reihenfolge ihrer Startzeiten. Die markierten Jobs werden an diese Jobs in beliebiger Sortierung angefügt.

Um die Laufzeit von GREEDY-SCHEDULING zu bestimmen, müssen wir noch etwas genauer auf 2. eingehen. Sind die ersten $i-1$ Jobs in 2. bereits abgearbeitet worden und sind l davon Startzeiten zugewiesen worden, so speichern wir diese l Startzeiten $t_1, \ldots, t_l$ in aufsteigend sortierter Reihenfolge in einem Array. Ist $t_1 \neq 0$, so setzen wir ausserdem $t_0 = 0$. Dann entscheiden wir, ob es ein j zwischen 1 und l gibt, so dass

1. $d_i \geq t_j$
2. $t_j - t_{j-1} \geq 2$

Existiert ein solches j, wählen wir das maximale j mit dieser Eigenschaft und setzen $t_{l+1} = t_j - 1$. Existiert kein solches j, so wird Job j_i markiert.
Mit dieser Vorgehensweise können wir in Algorothmus GREEDY-SCHEDULING für jeden Job in Zeit $\mathcal{O}(n)$ entscheiden, ob ihm eine Startzeit vor seiner Deadline zugewiesen wird und welches diese gegebenenfalls ist. Damit ist dann die Gesamtlaufzeit von GREEDY-SCHEDULING bei n Jobs $\mathcal{O}(n^2)$.
Nun zur Optimalität von GREEDY-SCHEDULING. Mit GREEDY bezeichnen wir die von Algorithmus GREEDY-SCHEDULING berechnete Reihenfolge, in der die Jobs auseführt werden. Angenommen, es gibt nun eine Reihenfolge OPTIMAL, die eine echt größere Gesamtbelohnung erreicht als GREEDY. Dann muss es einen Zeitpunkt t geben, zu dem in OPTIMAL ein Job j mit Deadline $d < t$ und Belohnung b gestartet wird, der in GREEDY nicht vor seiner Deadline gestartet wird. Ausserdem wird zum Zeitpunkt t in GREEDY ein Job j' mit Belohnug $b' < b$ gestartet. Dann aber hätte GREEDY-SCHEDULING den Job j ebenfalls vor seiner Deadline starten lassen müssen. Denn j hat größere Belohnung als j', wird daher vor j' im 2. Schritt von GREEDY-SCHEDULING betrachtet. Zu diesm Zeitpunkt war aber dann Zeitpunkt t als Startzeit noch frei. Spätestens zu diesem Zeitpunkt hätte GREEDY-SCHEDULING Job j vor seiner Deadline d gestartet.

AUFGABE 4:
Ein Teilgraph $G_T = (G, T)$ ist genaun dann azyklisch, wenn er ein Wald auf den Knoten von G ist. Wir hatten aber in der Vorlesung gesehen, dass für jeden Grapen G das Paar (V, I) mit

$$T \in I \text{ genau dann, wenn für } T \subseteq E \text{ der Graph } G_T = (G, T) \text{ ein Wald ist,}$$

ein Matroid ist. Damit ist auch für jeden Graphen das in der Aufgabe definierten Paare $(S_G, \mathcal{I})$ ein Matroid.