2. Aufgabenblatt zur Vorlesung

Entwurf und Analyse von Algorithmen

(Günter Rote, Astrid Kaffanke, Christian Knauer)

Abgabe: 2. November 2000

Hinweis: Die Vorlesung fällt am 9. und am 24. November aus, die Übungen finden jedoch planmäßig statt.

1. **Solarauto bei schlechtem Wetter** (6 Punkte).

   Frau X möchte mit ihrem Solarauto von Hamburg nach München fahren. Wenn keine Sonne scheint, kann an geeigneten Stellen der Akku mit Strom „aufgetankt" werden. Die Verbindung geht über Autobahnen direkt, die Solartankstellen auf der Strecke und die Entfernungen (in Straßenkilometern) zwischen ihnen sind auf der Landkarte angegeben. Wenn der Akku voll ist, reicht er zum Fahren von n Kilometern.

   Da schlechtes Wetter mit Dauerregen angesagt ist, möchte Frau X herausfinden, an welchen Tankstellen sie halten sollte, um *möglichst wenige Tankstopps* einzulegen.

   Gib einen effizienten Algorithmus an, der Frau X hilft, die richtigen Tankstellen auszuwählen. Begründe, warum der Algorithmus eine optimale Lösung liefert, und schätze seine Laufzeit ab.

2. **Raumverteilung** (freiwillig).

   Eine Aktivität A_i ist ein zeitlich beschränktes Ereignis mit einer Startzeit s_i und einer Stoppzeit f_i, wobei $s_i < f_i$. Zwei Aktivitäten A_i und A_j sind kompatibel, falls deren Zeitintervalle sich nicht überlappen, d.h. $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.

   Angenommen, es muss eine Menge von n Aktivitäten auf eine Anzahl von Räumen kompatibel verteilt werden, so dass möglichst wenige Räume benötigt werden. Überlege Dir, welches Entwurfsprinzip hier angewendet werden kann. Gib dann einen effizienten Algorithmus an, der berechnet, welche Aktivität in welchem Raum stattfinden soll. Begründe, warum der Algorithmus eine optimale Lösung liefert und schätze seine Laufzeit ab.

3. **Gebrochenes Rucksackproblem** (3 Punkte).

   (a) Was hat das System

   $$\text{maximiere} \quad \sum_{i=1}^{n} w_i x_i \quad \text{unter} \quad \sum_{i=1}^{n} g_i x_i \le G; \quad x_i \in \{0,1\}$$

   mit dem Rucksackproblem zu tun? Was bedeuten die Variablen x_i?

   (b) Welche Bedeutung hat es, wenn die letzte Bedingung durch $x_i \in [0,1]$ ersetzt wird?

   (c) Zeige, dass der Greedy-Algorithmus das in der Vorlesung vorgestellte *gebrochene Rucksackproblem* optimal löst.

Bitte wenden!

4. **Branch and bound**[1] (10 Punkte).

Praktische Aufgabe! Abgabe bis zum 9.9.2000

Programmiere das in der Vorlesung vorgestellte Verfahren zur Lösung des Rucksackproblems durch das systematische Durchsuchen von Lösungsbäumen. Verwende die in der Vorlesung vorgestellte Schranke, um frühzeitig Teile des Lösungsbaumes abzuschneiden. Verwende die Lösung des gebrochenen Rucksackproblems für die verbleibenden Gegenstände $x_{i+1}, \ldots, x_n$ als alternative Schranke und vergleiche die Wirkung der beiden Schranken miteinander.

Experimentiere mit zwei verschiedenen Strategien bezüglich der Suchreihenfolge, und mit 3 verschiedenen Eingabegrößen n. Dabei soll das größte Problem so gewählt sein, dass die Rechenzeit ca. 10 Sekunden beträgt.

Lass deinen Algorithmus mit 10 verschiedenen Beispielen aus jeder Klasse laufen. Die Beispieldaten sollen zufällig erzeugt werden. Protokolliere dabei die Anzahl der insgesamt besuchten Knoten des Lösungsbaumes, die Anzahl der besuchten Knoten bis zum Auffinden des Optimums und die Laufzeit des Algorithmus.

Schriftlich abzugeben sind eine Beschreibung, nach welcher Methode die Beispieldaten erzeugt wurden und eine Beschreibung der zwei verwendeten Suchstrategien. Außerdem sollen die Ergebnisse der Tests erläutert werden.

Per email abzugeben sind übersetzbarer Quellcode und die Beispieldaten in einer Form, die es uns erlaubt, das Programm mit diesen Daten zu testen.

5. **Längste aufsteigende Teilfolge** (freiwillig).

Gegeben sei eine Folge $\mathbf{a} = (a_1, \ldots, a_n)$ von n natürlichen Zahlen. Gesucht ist $lis(\mathbf{a})$, die Länge der längsten aufsteigenden Teilfolge von $\mathbf{a}$. Gib ein Verfahren an, das $lis(\mathbf{a})$ in $O(n^2)$ Zeit berechnet.

Hinweis für eine mögliche Lösung: Berechne zunächst für $1 \leq k \leq n$ die Länge der längsten aufsteigenden Teilfolge von $\mathbf{a}_k = (a_1, \ldots, a_k)$, deren letztes Element gerade a_k ist.

6. **Matrixmultiplikation** (7 Punkte).

In der Vorlesung wurde ein schneller Algorithmus vorgestellt, um $2^k \times 2^k$–Matrizen zu multiplizieren.

(a) Modifiziere das in der Vorlesung vorgestellte Verfahren so, dass man damit $n \times n$ Matrizen multiplizieren kann. Wie ist die Laufzeit deines Verfahrens, wenn der schnelle Algorithmus aus der Vorlesung zwei $n \times n$ Matrizen mit $n = 2^k$ mit $O(n^\alpha)$ Multiplikationen (für ein $\alpha < 3$) multiplizieren kann?

(b) Modifiziere deinen Algorithmus aus der ersten Teilaufgabe so, dass Du damit eine $m \times n$ Matrix mit einer $n \times k$ Matrix multiplizieren kannst. Leite wieder die Laufzeit deines Verfahrens her.

[1] Implementierungen im Rahmen der praktischen Übungsaufgaben müssen in der Programmiersprache Java erstellt werden. Die Abgabe dieser Aufgaben erfolgt per email an den jeweiligen Tutor. Dabei bitte nur lauffähigen und compilierbaren Quellcode schicken. Besteht die Implementierung aus mehreren Dateien, sind diese — unter Beibehaltung der Verzeichnisstruktur — in ein Archiv (tar,tgz,zip) zu verpacken.