Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Algorithmen und Datenstrukturen II
Große Übung #2

Arne Schmidt

03.05.2016

Probleme

Bisher kennengelernte Probleme:

- *(0-1-)Knapsack*
- *Maximum Knapsack*
- *Fractional Knapsack*
- *Integer Knapsack*
- *Subset Sum*
- *(Maximum Subset Sum) Problem 1.7*
- *Partition*

Was haben (fast) alle gemeinsam?

Reduktionen

Ein Problem P kann manchmal auf ein andere Problem P' reduziert werden, d.h., besitzen wir einen Algorithmus A', der P' löst, können wir einen Algorithmus A konstruieren, der A' benutzt und P löst.

Beispiel 1:
Reduktion von PARTITION auf KNAPSACK.

Das gab es in der Hausaufgabe!

Reduktionen

Beispiel 2:
Reduktion von INTEGER KNAPSACK auf MAXIMUM KNAPSACK.

Mögliche Reduktion:
Erstelle $\frac{Z}{z_i}$ Kopien von Objekt i.
Löse mit den Kopien MAXIMUM KNAPSACK.

Korrektheit ist schnell zu sehen.

Problem: Sehr viele Kopien von Objekt i; nämlich exponentiell viele!
D.h. unser Algorithmus besitzt exponentielle Laufzeit!

Polynomialzeit Reduktionen

Daher:
Konstruiere eine Reduktion, sodass die Laufzeit polynomiell in der Eingabegröße ist (ohne Algorithmus A').

Das hat zur Folge:
Besitzt A' polynomielle Laufzeit, hat auch A polynomielle Laufzeit!

Wir sagen: P ist höchstens so schwer wie P', falls eine polynomielle Reduktion von P auf P' existiert. Oder kurz: $P \leq P'$

Solche Reduktionen muss es nicht immer geben!

Eine dritte Reduktion?

Beispiel 3:
Reduktion von MAXIMUM KNAPSACK auf FRACTIONAL KNAPSACK.

Bekannt: FRACTIONAL KNAPSACK lässt sich effizient, d.h. in polynomieller Zeit, lösen (z.B. mit Greedy).

Bei MAXIMUM KNAPSACK ist es unbekannt! Gäbe es eine polynomielle Reduktion, so wäre MAXIMUM KNAPSACK effizient lösbar!

Unsere Probleme sind schwer!

Wir können zeigen:
Jedes bisher betrachtete Problem P (außer FRACTIONAL KNAPSACK) lässt sich auf ein anderes solches Problem P' reduzieren, also $P \leq P'$.
D.h. diese Probleme sind gleich schwer!

Für diese Probleme wird vermutet, dass kein effizienter Algorithmus existiert. (Siehe auch: "P vs NP")

Wie lösen wir diese Probleme?

Lösungsansätze

In der Vorlesung:

- *Dynamic Programming*
- *Branch and Bound*
- *Approximationen*
- *Heuristiken*

In der Übung: Exkurs zu linearer Optimierung.

Linear Programming

Ein lineares Programm ist wie folgt aufgebaut:

$\min / \max c^T x$	Zielfunktion/Was soll optimiert werden?
$Ax \leq b$	Nebenbedingungen/Welche Lösungen sind gültig?
$x \in \mathbb{R}^n$	Wertebereich der Variablen.

Solche linearen Programme können i.d.R. effizient gelöst werden.

Beispiel: Knapsack

Wir möchten den Wert maximieren. Also:

$$\max \sum_{i=1}^{n} x_i p_i$$

und der Rucksack darf nicht überfüllt werden:

$$\sum_{i=1}^{n} x_i z_i \leq Z$$

Wir dürfen Objekte nur ganz oder gar nicht benutzen, d.h. $x_i \in \{0, 1\}$. Achtung: Da x_i nur die Werte 0 und 1 annehmen kann, wird das lösen eines solchen lineares Programms schwer! Diese Art wird auch Integer Programm genannt.

Live-Demo mit CPLEX

Lösen von linearen Programmen

Die Black-Box CPLEX (oder andere Software) löst für uns lineare Programme. Wie geht das genau?
Diese Frage wird in der Vorlesung *Mathematische Methoden der Algorithmik*[1] beantwortet!

[1] MMA ist für Master-Studierende Inf/Winfo/IST im Wintersemester