



Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Proseminare WiSe 2010/11

Anmeldefrist

- Dienstag 01.06. Bis Donnerstag 10.06.
- Die Verteilung ist unabhängig von der Reihenfolge der Anmeldungen

Infos und Anmeldung

- <http://www.cs.uni-dortmund.de/proseminar>
- Dort steht alles weitere.

Organisatorisches

Testergebnisse

- In der nächsten Woche
- Einsicht im Rahmen der Übungen
- Bestehensquote >70%
- Inhaltlich: Induktion bei der Laufzeit

Organisatorisches

Übungen

- Änderung: Übungsabgabe bis Montag 12 Uhr möglich
- Dann allerdings keine Korrektur bis Dienstag
- Umfang der Übungen: DAP 2 gibt 12 Credits = 360 Arbeitsstunden pro Vorlesung = 24 pro Woche
- Damit ist ein Umfang von 8 Stunden pro zweiwöchentlichem Übungsblatt angemessen
- Im Vergleich zum Diplomstudiengang sind die Übungen leichter
- Diskussion: C in Übungen
- Webseiten

Dynamische Programmierung

Das Rucksackproblem

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Beispiel (siehe Vollversion)

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen in den Rucksack und haben Gesamtwert 13
- Objekt 2,3 und 4 passen und haben Gesamtwert 15

Dynamische Programmierung

Das Rucksackproblem

- Eingabe: Anzahl der Objekte n
Für jedes Objekt i seine ganzzahlige Größe $g[i]$ und seinen ganzzahligen Wert $v[i]$
Rucksackgröße W
- Ausgabe: $S \subseteq \{1, \dots, n\}$, so dass $\sum_{i \in S} g[i] \leq W$ und $\sum_{i \in S} v[i]$ maximal ist

Lösungsansatz

- Bestimme zunächst der Wert einer optimalen Lösung
- Leite dann die Lösung selbst aus der Tabelle des dynamischen Programms her

Dynamische Programmierung

Herleiten der Rekursion

- Sei $O \subseteq \{1, \dots, i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i$ und Rucksackgröße j
- Sei $\text{Opt}(k, j)$ der Wert einer optimalen Lösung für das Rucksackproblem mit Rucksackgröße j und den Objekten 1 bis k mit Größen $g[i]$ und Werten $v[i]$
- Gesucht: $\text{Opt}(n, W)$

Unterscheide, ob Objekt n in O ist:

- Fall 1 (Objekt i nicht in O):
 $\text{Opt}(i, j) = \text{Opt}(i-1, j)$
- Fall 2 (Objekt i in O):
 $\text{Opt}(i, j) = v[i] + \text{Opt}(i-1, j-g[i])$

Dynamische Programmierung

Rekursion

- Wenn $j < g[i]$ dann $\text{Opt}(i,j) = \text{Opt}(i-1,j)$
- Sonst,
 $\text{Opt}(i,j) = \max\{\text{Opt}(i-1,j), v[i] + \text{Opt}(i-1,j-g[i])\}$

Wenn Objekt i nicht in den Rucksack, sind in der optimalen Lösung nur Objekte aus $\{1, \dots, i-1\}$

Sonst ist entweder i in der optimalen Lösung oder die beste Lösung besteht aus Objekten aus $\{1, \dots, i-1\}$

Rekursionsabbruch

- $\text{Opt}(0,j) = 0$ für $0 \leq j \leq W$

Gibt es keine Objekte, so kann auch nichts in den Rucksack gepackt werden

Dynamische Programmierung

Rucksack(n,g,v,W)

1. **new array** Opt[0,...,n][0,...,W]
2. **for** j ← 0 **to** W **do**
3. Opt[0,j] ← 0
4. **for** i ← 0 **to** n **do**
5. **for** j ← 0 **to** W **do**
6. Berechne Opt[i,j] nach Rekursion
7. **return** Opt[n,W]

Laufzeit

- $O(nW)$

Dynamische Programmierung

Beispiel (siehe Vollversion)

Optimaler Lösungswert für $W=8$

n	0	2	3	5	7	9	10	12	13
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

	Größe	Wert
	g	v
1	5	2
2	3	4
	1	1
	2	3
	1	2
	7	3
	4	7
n	3	3

Dynamische Programmierung

Beobachtung:

- Sei R der Wert einer optimalen Lösung für die Elemente $1, \dots, i$
- Falls $g[i] \leq j$ und $\text{Opt}[i-1, j-g[i]] + v[i] = R$, so ist Objekt i in mindestens einer optimalen Lösung enthalten

Wie kann man eine optimale Lösung berechnen?

- Idee: Verwende Tabelle der dynamischen Programmierung
- Fallunterscheidung + Rekursion:
 - Falls das i -te Objekt in einer optimalen Lösung für Objekte 1 bis i und Rucksackgröße j ist, so gib es aus und fahre rekursiv mit Objekt $i-1$ und Rucksackgröße $W-g[i]$ fort
 - Ansonsten fahre mit Objekt $i-1$ und Rucksackgröße j fort

Dynamische Programmierung

RucksackLösung(Opt,i,j,g,v,K)

1. **if** $i=0$ **return** \emptyset
2. **else if** $g[i]>j$ **then return** RucksackLösung(Opt,i-1,j,g,v,K)
3. **else if** $K=v[i] + \text{Opt}[i-1,j-g[i]]$ **then**
 return $\{i\} \cup \text{RucksackLösung}(\text{Opt},i-1,j-g[i],g,v,K-v[i])$
4. **else return** RucksackLösung(Opt,i-1,j,g,v,K)

Lemma 28 (Beweis später)

- Hat die optimale Lösung für Objekte $1,\dots,i$ und Rucksackgröße j den Wert K , so berechnet Algorithmus Rucksacklösung eine Teilmenge S von $\{1,\dots,i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = K$ ist.

Dynamische Programmierung

RucksackLösung(Opt,i,j,g,v,K)

1. **if** $i=0$ **return** \emptyset
2. **else if** $g[i]>j$ **then return** *RucksackLösung*(Opt,i-1,j,g,v,K)
3. **else if** $K=v[i] + \text{Opt}[i-1,j-g[i]]$ **then**
 return $\{i\} \cup \text{RucksackLösung}(\text{Opt},i-1,j-g[i],g,v,K-v[i])$
4. **else return** *RucksackLösung*(Opt,i-1,j,g,v,K)

Aufruf

- Nach der Berechnung der Tabelle Opt von Rucksack wird *RucksackLösung* mit Opt, $i=n$, $j=W$, g, v, und $K=\text{Opt}[n,W]$ aufgerufen.
- Nach dem Lemma wird dann die optimale Lösung konstruiert

Dynamische Programmierung

Beispiel (siehe Vollversion)

K=0, j=0, i=0:
Es gilt i=0

n	0	2	3	5	7	9	10	12	13
	0	2	3	5	7	9	10	12	13
	0	2	3	5	6	7	9	10	10
	0	2	3	5	6	7	9	10	10
	0	1	3	4	5	7	8	8	8
	0	1	1	4	5	5	5	5	6
	0	0	0	4	4	4	4	4	6
1	0	0	0	0	0	2	2	2	2
0	0	0	0	0	0	0	0	0	0
	0	1							W

		Wert	
		g	v
1		5	2
2		3	4
		1	1
		2	3
		1	2
		7	3
		4	7
n		3	3

Dynamische Programmierung

Lemma 29

Nach dem k -ten Durchlauf der äußeren **for**-Schleife von `Rucksack(n,g,v,W)` gilt für alle $0 \leq i \leq k$ und $0 \leq j \leq W$, dass Feld `Opt[i,j]` den Wert einer optimalen Lösung für Objekte 1 bis i mit Größen $g[i]$ und Werten $v[i]$ und Rucksackgröße j .

Beweis:

- Wir beweisen die Aussage per Induktion über k .
- (I.A.) $k=0$: Vor dem ersten Schleifendurchlauf wurde `Opt[0,j]` für alle $0 \leq j \leq W$ mit 0 initialisiert. Da keine Objekte vorhanden sind, können auch keine mitgenommen werden. Daher stimmt die Aussage.
- (I.V.) Die Aussage stimmt für $k-1$.

Dynamische Programmierung

Lemma 29

Nach dem k -ten Durchlauf der äußeren **for**-Schleife von `Rucksack(n,g,v,W)` gilt für alle $0 \leq i \leq k$ und $0 \leq j \leq W$, dass Feld `Opt[i,j]` den Wert einer optimalen Lösung für Objekte 1 bis i mit Größen $g[i]$ und Werten $v[i]$ und Rucksackgröße j .

Beweis:

- (I.V.) Die Aussage stimmt für $k-1$.
- (I.S.) Wir müssen zeigen, dass `Opt[k,j]` genau den Wert einer optimalen Lösung für Objekte 1 bis k bei Rucksackgröße j hat. Sei dazu R der Wert einer optimalen Lösung. Wir zeigen $R \leq \text{Opt}[k,j]$ und $R \geq \text{Opt}[k,j]$. Damit gilt $\text{Opt}[k,j] = R$.
- „ $R \leq \text{Opt}[k,j]$ “: Sei $O \subseteq \{1, \dots, k\}$ eine optimale Lösung für Rucksackgröße j . Ist $k \in O$, so ist $g[k] \leq j$ und $\text{Opt}[k,j] \geq v[k] + \text{Opt}[k-1, j-g[k]] = R$ nach (I.V.). Ist $k \notin O$, so ist $\text{Opt}[k,j] = \text{Opt}[k-1, j] = R$.

Dynamische Programmierung

Lemma 29

Nach dem k -ten Durchlauf der äußeren **for**-Schleife von `Rucksack(n,g,v,W)` gilt für alle $0 \leq i \leq k$ und $0 \leq j \leq W$, dass Feld `Opt[i,j]` den Wert einer optimalen Lösung für Objekte 1 bis i mit Größen $g[i]$ und Werten $v[i]$ und Rucksackgröße j .

Beweis:

- „ $R \geq \text{Opt}[k,j]$ “: Es gilt $\text{Opt}[k,j] = \max\{\text{Opt}[k-1,j], v[k] + \text{Opt}[k-1, j-g[k]]\}$ oder $\text{Opt}[k,j] = \text{Opt}[k-1,j]$. Wird das Maximum durch $\text{Opt}[k-1,j]$ erreicht und im zweiten Fall folgt $R \geq \text{Opt}[k,j] = \text{Opt}[k-1,j]$, weil $\text{Opt}[k-1,j]$ nach (I.V.) der Wert einer optimalen Lösung für Objekte $1, \dots, k-1$ ist, es somit also auch eine Teilmenge der Objekte $1, \dots, k$ gibt, die den Wert $\text{Opt}[k,j]$ hat. Wird das Maximum durch $v[k] + \text{Opt}[k-1, j-g[k]]$ erreicht, so folgt $R \geq \text{Opt}[k,j]$, da $v[k] + \text{Opt}[k-1, j-g[k]]$ den Kosten von $\{k\} \cup X$ entspricht, wobei X eine optimale Lösung der ersten $k-1$ Objekte bei Rucksackgröße $j-g[k]$ ist. Da es jeweils eine Lösung mit Kosten $\text{Opt}[k,j]$ gibt, folgt $R \geq \text{Opt}[k,j]$.

Dynamische Programmierung

Lemma 28

- Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert K , so berechnet Algorithmus Rucksacklösung eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = K$ ist.

Beweis:

- Beweis per Induktion über i .
- (I.A.) Ist $i=0$, so gibt der Algorithmus die leere Menge zurück. Dies ist korrekt, da kein Objekt in den Rucksack gepackt werden kann.
- (I.V.) Die Aussage stimmt für $i-1$.
- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus ruft in diesem Fall $\text{RucksackLösung}(\text{Opt}, i-1, j, g, v, K)$. Dies ist nach (I.V.) korrekt.

Dynamische Programmierung

Lemma 28

- Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert K , so berechnet Algorithmus Rucksacklösung eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = K$ ist.

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\text{RucksackLösung}(\text{Opt}, i-1, j, g, v, K)$ zurück. Dies ist nach (I.V.) korrekt. Ist $g[i] \leq j$ und $K = v[i] + \text{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält. In diesem Fall gibt der Algorithmus $\{i\} \cup \text{RucksackLösung}(\text{Opt}, i-1, j-g[i], g, v, K-v[i])$ zurück. Dies ist nach (I.V.) korrekt. Ist $g[i] \leq j$ und $K > v[i] + \text{Opt}[i-1, j-g[i]]$, so kann Objekt i nicht zu einer optimalen Lösung gehören. Der Algorithmus gibt in diesem Fall $\text{RucksackLösung}(A, i-1, j, g, v, K)$ zurück. Dies ist nach (I.V.) korrekt.

Dynamische Programmierung

RucksackKomplett(n, g, v, W)

1. Rucksack(n, g, v, W)
2. **return** RucksackLösung($Opt, n, W, g, v, Opt[n, W]$)

Satz 30

Algorithmus RucksackKomplett berechnet in $\Theta(nW)$ Zeit den Wert einer optimalen Lösung, wobei n die Anzahl der Objekte ist und W die Größe des Rucksacks.

Beweis:

- Die Laufzeit wird durch Algorithmus Rucksack definiert und ist somit $\Theta(nW)$. Die Korrektheit folgt aus den beiden Lemmas.