technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Vorgehensweise bei dynamischer Programmierung

1. Bestimme rekursive Struktur einer optimalen Lösung.

2. Entwirf rekursive Methode zur Bestimmung des Wertes einer optimalen Lösung.

3. Transformiere rekursive Methode in eine iterative (bottom-up) Methode zur Bestimmung des Wertes einer optimalen Lösung.

4. Bestimme aus dem Wert einer optimalen Lösung und den in 3. ebenfalls berechneten Zusatzinformationen eine optimale Lösung.

Dynamische Programmierung

Das Rucksackproblem

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

Dynamische Programmierung

Das Rucksackproblem

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

Beispiel

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

Dynamische Programmierung

Das Rucksackproblem

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

Beispiel

- 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

Dynamische Programmierung

Das Rucksackproblem

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

Beispiel

- 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: $S \subseteq \{1, ..., n\}$, so dass $\sum_{i \in S} g[i] \le W$ und $\sum_{i \in S} v[i]$ maximal ist

Dynamische Programmierung

Lösungsansatz

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

Dynamische Programmierung

Herleiten der Rekursion

- Sei $O \subseteq \{1, \ldots, i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \ldots, i$ und Rucksackgröße j
- Sei $\mathrm{Opt}(i, j)$ der *Wert* einer solchen optimalen Lösung
- Gesucht: $\mathrm{Opt}(n, W)$

Dynamische Programmierung

Aufgabe

Bestimmen Sie eine Rekursionsgleichung für $\mathrm{Opt}(i, j)$

Dynamische Programmierung

Lemma 24 (Struktur einer optimalen Lösung des Rucksackproblems)

- Sei $O \subseteq \{1, \dots, i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i$ und Rucksackgröße j. Es bezeichne $\mathrm{Opt}(i, j)$ den Wert dieser optimalen Lösung. Dann gilt:
- (a) Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$. Insbesondere gilt $\mathrm{Opt}(i, j) = v[i] + \mathrm{Opt}(i-1, j - g[i])$.
- (b) Ist Objekt i nicht in O enthalten, so ist O eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße j. Insbesondere gilt $\mathrm{Opt}(i, j) = \mathrm{Opt}(i-1, j)$.

Dynamische Programmierung

Beweis

- (a) z.z.: Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \ldots, i-1$ und Rucksackgröße $j - g[i]$. Insbesondere gilt $\mathrm{Opt}(i, j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.

Dynamische Programmierung

Beweis

- (a) z.z.: Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$. Insbesondere gilt $\mathrm{Opt}(i,j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.
- Für $i = 1$ ist die Aussage offensichtlich korrekt. Sei also $i > 1$.

Dynamische Programmierung

Beweis

- (a) z.z.: Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \ldots, i-1$ und Rucksackgröße $j - g[i]$. Insbesondere gilt $\mathrm{Opt}(i,j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.
- Für $i = 1$ ist die Aussage offensichtlich korrekt. Sei also $i > 1$.
- Sei O eine optimale Lösung mit Wert $\mathrm{Opt}(i,j)$, die Objekt i enthält. Da Objekt i Größe $g[i]$ hat, gilt sicher, dass $O \setminus \{i\}$ eine Gesamtgröße von höchstens $j - g[i]$ hat. Damit ist $O \setminus \{i\}$ eine gültige Lösung für das Rucksackproblem mit Objekten $1, \ldots, i-1$ und Rucksackgröße $j - g[i]$.

Dynamische Programmierung

Beweis

- (a) z.z.: Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$. Insbesondere gilt $\mathrm{Opt}(i,j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.
- Für $i = 1$ ist die Aussage offensichtlich korrekt. Sei also $i > 1$.
- Sei O eine optimale Lösung mit Wert $\mathrm{Opt}(i,j)$, die Objekt i enthält. Da Objekt i Größe $g[i]$ hat, gilt sicher, dass $O \setminus \{i\}$ eine Gesamtgröße von höchstens $j - g[i]$ hat. Damit ist $O \setminus \{i\}$ eine gültige Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$.

Dynamische Programmierung

Beweis

- Annahme: $O \setminus \{i\}$ hat Wert $R = \mathrm{Opt}(i, j) - v[i]$ und ist keine optimale Lösung für das Rucksackproblem mit Objekten $1, \ldots, i-1$ und Rucksackgröße $j - g[i]$.

Dynamische Programmierung

Beweis

- Annahme: $O \setminus \{i\}$ hat Wert $R = \mathrm{Opt}(i,j) - v[i]$ und ist keine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$.
- Dann gibt es eine bessere Lösung O^* für dieses Problem mit Wert $R^* > R$. Weiterhin ist $O^* \cup \{i\}$ eine gültige Lösung für das Rucksackproblem mit Objekten $1, \dots, i$ und Rucksackgröße j. Der Wert dieser Lösung ist $R^* + v[i] > R + v[i] = \mathrm{Opt}(i,j)$. Widerspruch zur Optimalität von O.

Dynamische Programmierung

Beweis

- Annahme: $O \setminus \{i\}$ hat Wert $R = \mathrm{Opt}(i,j) - v[i]$ und ist keine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$.
- Dann gibt es eine bessere Lösung O^* für dieses Problem mit Wert $R^* > R$. Weiterhin ist $O^* \cup \{i\}$ eine gültige Lösung für das Rucksackproblem mit Objekten $1, \dots, i$ und Rucksackgröße j. Der Wert dieser Lösung ist $R^* + v[i] > R + v[i] = \mathrm{Opt}(i,j)$. Widerspruch zur Optimalität von O.
- Damit ergibt sich sofort $\mathrm{Opt}(i,j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.

Dynamische Programmierung

Beweis

- Annahme: $O \setminus \{i\}$ hat Wert $R = \mathrm{Opt}(i,j) - v[i]$ und ist keine optimale Lösung für das Rucksackproblem mit Objekten $1, \dots, i-1$ und Rucksackgröße $j - g[i]$.
- Dann gibt es eine bessere Lösung O^* für dieses Problem mit Wert $R^* > R$. Weiterhin ist $O^* \cup \{i\}$ eine gültige Lösung für das Rucksackproblem mit Objekten $1, \dots, i$ und Rucksackgröße j. Der Wert dieser Lösung ist $R^* + v[i] > R + v[i] = \mathrm{Opt}(i,j)$. Widerspruch zur Optimalität von O.
- Damit ergibt sich sofort $\mathrm{Opt}(i,j) = v[i] + \mathrm{Opt}(i-1, j-g[i])$.

Dynamische Programmierung

Beweis

(b) analog zu (a).

Dynamische Programmierung

Korollar 25 (Rekursion zur Berechnung der Kosten einer opt. Lösung)

Es gilt:

- $\mathrm{Opt}(0, j) = 0$ für $0 \le j \le W$,
- $\mathrm{Opt}(i, j) = \max\{\mathrm{Opt}(i-1, j), v[i] + \mathrm{Opt}(i-1, j-g[i])\}$, falls $i > 0$ und $g[i] \le j$,
- $\mathrm{Opt}(i, j) = \mathrm{Opt}(i-1, j)$, sonst.

Dynamische Programmierung

Korollar 25 (Rekursion zur Berechnung der Kosten einer opt. Lösung)

Es gilt:

- $\mathrm{Opt}(0, j) = 0$ für $0 \leq j \leq W$,
- $\mathrm{Opt}(i, j) = \max\{\mathrm{Opt}(i-1, j), v[i] + \mathrm{Opt}(i-1, j-g[i])\}$, falls $i > 0$ und $g[i] \leq j$,
- $\mathrm{Opt}(i, j) = \mathrm{Opt}(i-1, j)$, sonst.

Beweis

Aufgrund von Lemma 24 wissen wir, dass der Wert einer optimalen Lösung entweder durch $\mathrm{Opt}(i-1, j)$ oder durch $v[i] + \mathrm{Opt}(i-1, j-g[i])$ gegeben sind. Letzterer Fall kann nur auftreten, wenn $g[i] \leq j$ ist. Beide Werte entsprechen außerdem dem Wert einer zulässigen Lösung. Dies zeigt die Korrektheit der Rekursion.

Dynamische Programmierung

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

Rekursion

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

$$\mathrm{Opt}(i, j) = \max\{\mathrm{Opt}(i-1, j), v[i] + \mathrm{Opt}(i-1, j-g[i])\}$$

Rekursionsabbruch

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

Dynamische Programmierung

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

Rekursion

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

$$\mathrm{Opt}(i,j) = \max\{\mathrm{Opt}(i-1,j), v[i] + \mathrm{Opt}(i-1, j-g[i])\}$$

Rekursionsabbruch

- $\mathrm{Opt}(0,j) = 0$ für $0 \le j \le W$

Dynamische Programmierung

Rekursion

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

$$\mathrm{Opt}(i, j) = \max\{\mathrm{Opt}(i-1, j), v[i] + \mathrm{Opt}(i-1, j-g[i])\}$$

Rekursionsabbruch

- $\mathrm{Opt}(0, j) = 0$ für $0 \le j \le W$

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** $j \leftarrow 0$ **to** W **do**
3. Opt$[0, j] \leftarrow 0$
4. **for** $i \leftarrow 1$ **to** n **do**
5. **for** $j \leftarrow 0$ **to** W **do**
6. Berechne Opt$[i, j]$ nach Rekursion
7. **return** Opt$[n, W]$

Dynamische Programmierung

Rucksack(n, g, v, W)

new array Opt$[0..n][0..W]$
for $j \leftarrow 0$ **to** W **do**
    Opt$[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 0$ **to** W **do**
        Berechne Opt$[i, j]$ nach Rekursion
return Opt$[n, W]$

Laufzeit

$\mathbf{O}(nW)$

Dynamische Programmierung

Beispiel

n									
1									
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

Beispiel

n									
1	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0
	0	1							W

$g[1] > j$:
Also $\mathrm{Opt}[i,j] = \mathrm{Opt}[i-1,j]$

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

Beispiel

n									
1	0	0	0	0	0	2			
0	0	0	0	0	0	0	0	0	0
	0	1							W

$$\mathrm{Opt}[i,j] = \max\{\mathrm{Opt}[i-1,j], v[i] + \mathrm{Opt}[i-1, j-g[i]]\}$$

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

Beispiel

	0	1							W
n									
1	0	0	0	0	0	2	2		
0	0	0	0	0	0	0	0	0	0

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

Beispiel

n									
1	0	0	0	0	0	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

Beispiel

n									
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

Beispiel

n									
	0	0	0						
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

Beispiel

n									
	0	0	0	4	4				
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

Beispiel

n									
	0	0	0	4	4	4	4	4	
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

Beispiel

n									
	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

Beispiel

n									
	0								
	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

Beispiel

n									
	0	1	1						
	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 → g, Wert → v

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

Dynamische Programmierung

Beispiel

n									
	0	1	1	4					
	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

Beispiel

n									
	0	1	1	4	5	5	5	5	
	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

Beispiel

n									
	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

Beispiel

n									
	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

Beispiel

n									
	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

Beispiel

n									
	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

Beispiel

n									
	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

Beispiel

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

Optimaler Lösungswert für $W = 8$

Beispiel

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 $1, \ldots, i$
- Falls $g[i] \leq j$ und $\mathrm{Opt}[i-1, j-g[i]] + v[i] = R$, so ist Objekt i in mindestens einer optimalen Lösung enthalten

Dynamische Programmierung

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 $j-g[i]$ fort
  - Ansonsten fahre mit Objekt $i-1$ und Rucksackgröße j fort

Dynamische Programmierung

$\text{RucksackLösung}(\text{Opt}, g, v, i, j)$

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

Dynamische Programmierung

$\text{RucksackLösung}(\text{Opt}, g, v, i, j)$

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

Aufruf

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

Dynamische Programmierung

$\mathrm{Opt}[i,j] = 13, j = 8, i = 8$:
Es gilt $\mathrm{Opt}[i,j] > v[i] +$
$\mathrm{Opt}[i-1, j-g[i]]$

Beispiel

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

Beispiel

$\mathrm{Opt}[i,j] = 13, j = 8, i = 7$:
Es gilt $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$

	0	1							W
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

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

Beispiel

$\mathrm{Opt}[i,j] = 6, j = 4, i = 6$:
Es gilt $g[i] > j$

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

$\mathrm{Opt}[i,j] = 6, j = 4, i = 5$:
Es gilt $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}\big[i-1, j-g[i]\big]$

Beispiel

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

$\mathrm{Opt}[i,j] = 6, j = 4, i = 5$:
Es gilt $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}\big[i-1, j-g[i]\big]$

Beispiel

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

Beispiel

$\mathrm{Opt}[i,j] = 4, j = 3, i = 4$:
Es gilt $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}\big[i-1, j-g[i]\big]$

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

$\mathrm{Opt}[i,j] = 1, j = 1, i = 3$:
Es gilt $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}\big[i-1, j-g[i]\big]$

Beispiel

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	g 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

$\mathrm{Opt}[i,j] = 0, j = 0, i = 1$:
Es gilt $g[i] > j$

Beispiel

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

Beispiel

$\mathrm{Opt}[i,j] = 0, j = 0, i = 1$:
Es gilt $g[i] > j$

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

$\mathrm{Opt}[i,j] = 0, j = 0, i = 0$:
Es gilt $i = 0$

Beispiel

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

Lemma 26

Hat die optimale Lösung für Objekte $1, \ldots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \ldots, i\}$, so dass $\sum_{i \in S} g[i] \le j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \ldots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \ldots, i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Beweis:

- Aufgrund von Korollar 25 enthält $\mathrm{Opt}[i,j]$ jeweils den Wert $\mathrm{Opt}(i,j)$ einer optimalen Lösung für Objekte $\{1, \ldots, i\}$ und Rucksackgröße j. Wir zeigen das Lemma per Induktion.

Dynamische Programmierung

Lemma 26

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

Beweis:

- Aufgrund von Korollar 25 enthält $\text{Opt}[i, j]$ jeweils den Wert $\text{Opt}(i, j)$ einer optimalen Lösung für Objekte $\{1, \dots, i\}$ und Rucksackgröße j. Wir zeigen das Lemma per Induktion.
- 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.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i \in S} g[i] \le j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Beweis:

- Aufgrund von Korollar 25 enthält $\mathrm{Opt}[i,j]$ jeweils den Wert $\mathrm{Opt}(i,j)$ einer optimalen Lösung für Objekte $\{1, \dots, i\}$ und Rucksackgröße j. Wir zeigen das Lemma per Induktion.
- 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 $i - 1$.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i, j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i \in S} g[i] \le j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i, j)$ ist.

Beweis:

- Aufgrund von Korollar 25 enthält $\mathrm{Opt}[i, j]$ jeweils den Wert $\mathrm{Opt}(i, j)$ einer optimalen Lösung für Objekte $\{1, \dots, i\}$ und Rucksackgröße j. Wir zeigen das Lemma per Induktion.
- 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 $i - 1$.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \ldots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i, j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \ldots, i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i, j)$ ist.

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \ldots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \ldots, i\}$, so dass $\sum_{i \in S} g[i] \leq j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.
- Ist $g[i] \leq j$ und $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält.

Dynamische Programmierung

Lemma 26

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

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.
- Ist $g[i] \leq j$ und $\mathrm{Opt}[i, j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält. In diesem Fall gibt der Algorithmus $\{i\} \cup \mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j-g[i])$ zurück. Dies ist nach (I.V.) korrekt.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i\in S} g[i] \le j$ und $\sum_{i\in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.
- Ist $g[i] \le j$ und $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält. In diesem Fall gibt der Algorithmus $\{i\} \cup$ $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j-g[i])$ zurück. Dies ist nach (I.V.) korrekt.
- Ist $g[i] \le j$ und $\mathrm{Opt}[i,j] > v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so kann Objekt i nicht zu einer optimalen Lösung gehören.

Dynamische Programmierung

Lemma 26

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

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.
- Ist $g[i] \leq j$ und $\mathrm{Opt}[i, j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält. In diesem Fall gibt der Algorithmus $\{i\} \cup \mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j-g[i])$ zurück. Dies ist nach (I.V.) korrekt.
- Ist $g[i] \leq j$ und $\mathrm{Opt}[i, j] > v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so kann Objekt i nicht zu einer optimalen Lösung gehören. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) korrekt.

Dynamische Programmierung

Lemma 26

Hat die optimale Lösung für Objekte $1, \dots, i$ und Rucksackgröße j den Wert $\mathrm{Opt}(i,j)$, so berechnet Algorithmus $\mathrm{RucksackLösung}$ eine Teilmenge S von $\{1, \dots, i\}$, so dass $\sum_{i \in S} g[i] \le j$ und $\sum_{i \in S} v[i] = \mathrm{Opt}(i,j)$ ist.

Beweis:

- (I.S.) Ist $g[i] > j$, so kann Objekt i Teil keiner Lösung sein. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) und Lemma 24 korrekt.
- Ist $g[i] \le j$ und $\mathrm{Opt}[i,j] = v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so gibt es eine optimale Lösung, die Objekt i enthält. In diesem Fall gibt der Algorithmus $\{i\} \cup \mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j-g[i])$ zurück. Dies ist nach (I.V.) korrekt.
- Ist $g[i] \le j$ und $\mathrm{Opt}[i,j] > v[i] + \mathrm{Opt}[i-1, j-g[i]]$, so kann Objekt i nicht zu einer optimalen Lösung gehören. Der Algorithmus gibt in diesem Fall $\mathrm{RucksackLösung}(\mathrm{Opt}, g, v, i-1, j)$ zurück. Dies ist nach (I.V.) korrekt.

Dynamische Programmierung

$\text{RucksackKomplett}(n, g, v, W)$

1. $\text{Rucksack}(n, g, v, W)$
2. $\textbf{return}\ \text{RucksackLösung}(\text{Opt}, g, v, n, W)$

Dynamische Programmierung

Satz 27

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

Beweis:

- Die Laufzeit von Algorithmus $\mathrm{Rucksacklösung}$ ist $\mathbf{\Theta}(n)$, da sich bei jedem rekursiven Aufruf der erste Parameter um 1 reduziert, es nur jeweils einen rekursiven Aufruf gibt und jeder Aufruf konstante Zeit benötigt.
- Die Laufzeit wird durch Algorithmus $\mathrm{Rucksack}$ dominiert und ist somit $\mathbf{\Theta}(nW)$. Die Korrektheit folgt aus den beiden Lemmas.