# Algorithmen & Datenstrukturen

Herbst 2021

Vorlesung 7

Dynamisches Programmieren, Teil 2

# Dynamisches Programmieren (DP)

Allgemeine Vorgehensweise:

1. Designe Induktion / Rekurrenz
2. Definiere Tabelle (Dimensionen, Initialwerte)
3. Fülle Bottom-up
4. Lösung durch Zurückverfolgen

## Teilsummenproblem (subset sum)

gegeben: Geschenke von verschiedenen Wert

$$7, 105, 11, 19, 5, \ldots$$

Teile gerecht zwischen zwei Geschwistern, wenn es geht

Allgemeiner:

gegeben: $A[1], \ldots, A[n]$ und $b \in \mathbb{N}$

gesucht: $I \subseteq \{1, \ldots, n\}$ so daß

$b$ ist dann Teilsumme von $A$ $\quad \sum_{i \in I} A[i] = b$ falls möglich

Beispiel: $A$: 5, 3, 7, 3, 1

| $b$: | |
|---|---|
| 9 | ja |
| 2 | nein |
| 17 | nein |
| $> 19$ | nein |

Naiver Algorithmus: probiere alle Teilmengen

$O(2^n \cdot n)$ Laufzeit

DP: Grundidee:

$s$ ist Teilsumme von $A[1..n]$ $\qquad s = \sum_{i \in I} A[i]$

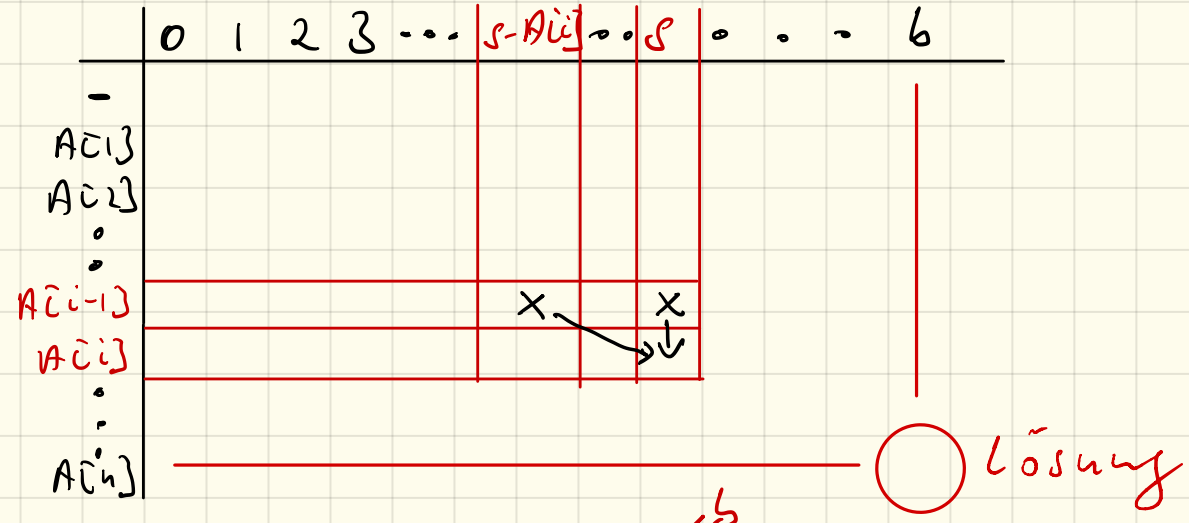
$\Rightarrow$ $s$ ist Teilsumme von $A[1..n-1]$ $\qquad n \notin I$

oder $s - A[n]$ ist Teilsumme von $A[1..n-1]$ $\qquad n \in I$

$TS(i,s)$: Wahrheitswert (also 1 oder 0) von "$s$ ist Teilsumme von $A[1..i]$"

Rekurrenz: $TS(i,s) = TS(i-1,s) \vee TS(i-1, s-A[i])$ (∨ = oder)

Tabelle:




Beispiel:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 (=b) |
|---|---|---|---|---|---|---|---|---|---|---|
| - | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 7 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 ✓ |

A

Zurückverfolgen: Jeder Sprung nach links gibt eine Zahl: 5, 3, 1 (nicht eindeutig)

neu: Laufzeit hängt von Größe einer Zahl ab!

Laufzeit: $O(bn)$ ← Verhalten für $n \to \infty$, $b \to \infty$

Speicher: $O(bn)$

z.B. alle $n$ Zahlen in $A$ sind 64 bit, aber $b$ kann beliebig groß sein, braucht $\log_2 b$ bit

$\Rightarrow$ Eingabegröße ist nicht mehr $n$, sondern $\Theta(n + \log(b))$

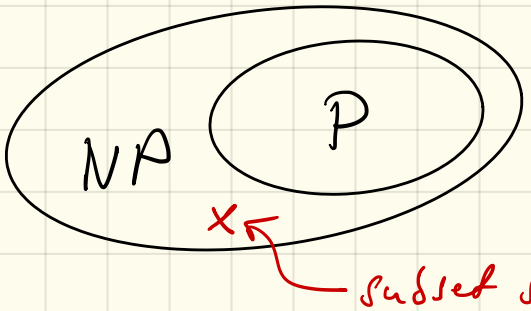
$b = 2^n$: Eingabe $\Theta(n)$, Laufzeit $O(n 2^n)$ exponentiell

$b = n^c$: Eingabe $\Theta(n)$, Laufzeit $O(n^{c+1})$ polynomiell

Man sagt: Laufzeit ist pseudopolynomiell

P: Menge aller Probleme mit polynomieller Laufzeit

NP: Menge aller Probleme bei denen man in polynomieller Zeit testen kann ob eine Lösung korrekt ist




subset sum

Vermutung: $P \neq NP$

Beweis $\Longrightarrow$ sehr berühmt

Bemerkung: in P, NP betrachtet man Entscheidungsprobleme (z.B. "ist $b$ Teilsumme von $A$?")

# Rucksackproblem (knapsack problem)

gegeben: • Rucksack mit Gewichtslimit $W$
• $n$ Gegenstände mit Gewicht $w_i \in \mathbb{N}$, Wert $v_i \in \mathbb{N}$, $i = 1..n$

gesucht: $I \subseteq \{1..n\}$ so daß $\sum_{i \in I} w_i \leq W$
und $\sum_{i \in I} v_i$ maximal

Naive Algorithmus: alle $I$ ausprobieren, jedes nehmen: Laufzeit $O(2^n n)$

"Greedy" Algorithmus: sortiere Gegenstände nach Wertdichte $v_i / w_i$, wähle in dieser Reihenfolge.

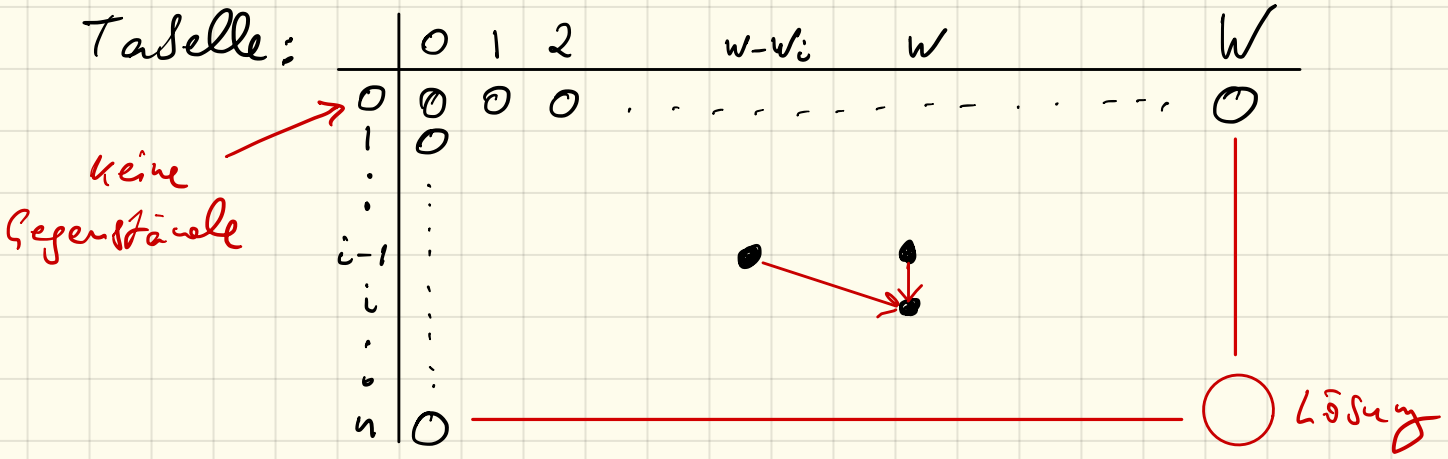
Kann beliebig schlecht sein.
Beispiel: $(v_1, w_1) = (1, 1)$ $(v_2, w_2) = (W-1, W)$

DP
Optimale Lösung enthält $n$ oder nicht
⇒ optimale Lösung für $n$ Gegenstände mit $W$
ist opt. Lösung für $n-1$ " mit $W$
oder " für $n-1$ " mit $W - w_n$

$MV(i, w)$ = Max. Wert von $I \subseteq \{1..i\}$ mit Schranke $W$
↑
Max value

Rekurrenz: $MV(i,w) = \max(MV(i-1,w),\ MV(i-1,w-w_i)+v_i)$

Tabelle:

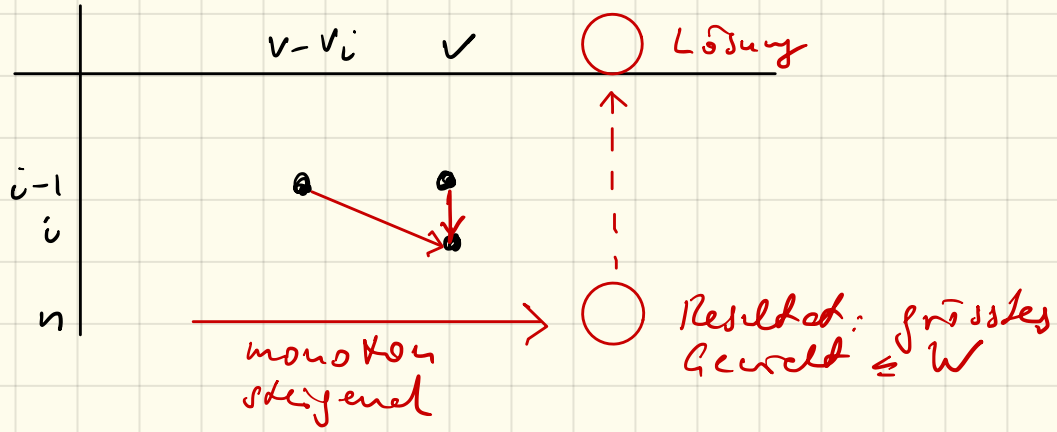



Laufzeit/Speicher: $O(n\,W)$ (pseudopolynomiell)

Geht auch mit $O(n\,V)$, $V = v_1 + \ldots + v_n$:

$MinG(i,v)$ = minimales Gewicht um mit $I \subseteq \{1..i\}$ Wert $\geq v$ zu erhalten

$MinG(i,v) = \min(MinG(i-1,v),\ MinG(i-1,v-v_i)+w_i)$




Wie können wir das beschleunigen?

Neue Idee: Berechnung einer approximativen Lösung.

Input bisher: $w_i, v_i, W$ $\quad$ $K \in \mathbb{N}$ $\quad$ INPUT

Input approx: $w_i, \lfloor v_i/K \rfloor, W$ $\quad$ $\overline{\text{INPUT}}$ ← approximiert

Beispiel: Werte 112, 78, 1001, 17, 237, .....

$K=10$, $\overline{\text{Werte}}$ 11, 7, 100, 1, 23, .....

$\text{INPUT} \xrightarrow{O(nV)} OPT \subseteq \{1..n\}$

$\overline{\text{INPUT}} \xrightarrow{O(n\bar{V})} \overline{OPT} \subseteq \{1..n\}$ $\quad$ $\bar{V}$ ist $\approx K$ mal kleiner ⇒ Algo ist $\approx K$ mal schneller

$O(n\bar{V}) \subseteq O(n^2 v_{max}/K)$

denn: $\bar{V} = \sum_{i=1}^{n} \lfloor \frac{v_i}{K} \rfloor \leq \sum_{i=1}^{n} \frac{v_i}{K} \leq \frac{1}{K} \sum_{i=1}^{n} v_i \leq \frac{n}{K} v_{max}$ ($v_{max}$: maximales $v_i$)

$Wert(OPT) = \sum_{i \in OPT} v_i$

$Wert(\overline{OPT}) = \sum_{i \in \overline{OPT}} v_i$ $\quad$ } wie weit voneinander entfernt

$\varepsilon = \varepsilon(K)$ dann wählbar um beliebig gut zu approximieren

Ziel: $Wert(\overline{OPT}) \geq (1-\varepsilon)\, Wert(OPT)$

Es gilt: $\frac{v_i}{K} - 1 \leq \lfloor \frac{v_i}{K} \rfloor \leq \frac{v_i}{K} \iff v_i - K \leq K \lfloor \frac{v_i}{K} \rfloor \leq v_i$

$$\sum_{i \in OPT} (v_i - K) \leq \sum_{i \in OPT} K \lfloor \frac{v_i}{K} \rfloor$$

$$\leq K \sum_{i \in \overline{OPT}} \lfloor \frac{v_i}{K} \rfloor$$ (Optimalität von $\overline{OPT}$ für Gewichte $\lfloor v_i/K \rfloor$)

$$\leq \sum_{i \in \overline{OPT}} v_i$$

$$= Wert(\overline{OPT})$$

$$\sum_{i \in OPT} (v_i - K) = Wert(OPT) - \sum_{i \in OPT} K$$

$$\geq Wert(OPT) - nK$$

also: $Wert(\overline{OPT}) \geq Wert(OPT) - nK \overset{!}{\geq} (1-\varepsilon)\, Wert(OPT)$ ← soll sein

heißt: $-nK \overset{!}{\geq} -\varepsilon\, Wert(OPT)$

$\Leftrightarrow \quad K \overset{!}{\leq} \frac{\varepsilon}{n} Wert(OPT)$

Annahme: alle $w_i \leq W$ (sonst entferne diese Gegenstände in $O(n)$)

$\Rightarrow Wert(OPT) \geq v_{max}$ — erfüllt dann

also wähle $K = \frac{\varepsilon}{n} v_{max} \leq \frac{\varepsilon}{n} Wert(OPT)$

$\Rightarrow$ Laufzeit $O(n^3/\varepsilon)$ polynomiell in $n$ und $1/\varepsilon$

"Fully polynomial time approximation scheme"