

Algorithmen & Datenstrukturen

Herbst 2022

Vorlesung 7

Dynamisches Programmieren, Teil 2

Dynamisches Programmieren (DP)

Allgemeine Vorgehensweise:

1. Designe Inklusion/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, ...

Teile gerecht zwischen zwei Geschwister,
wenn es geht

Allgemeiner:

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

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

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

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

$b = 9$	ja
2	nein
17	nein
> 19	nein

Naiver Algorithmus: probiere alle Teilsummen
 $O(2^n \cdot n)$ teuer

DP: Grundidee:

b ist Teilsumme von $A[1..n]$
 $\Rightarrow b$ ist Teilsumme von $A[1..n-1]$
 oder $b - A[n]$ ist Teilsumme von $A[1..n-1]$

$b = \sum_{i \in I} A[i]$
 $n \notin I$
 $n \in I$

$TS(i, s)$: Wahrheitsgehalt (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:

	0	1	2	3	...	$s - A[i]$	s	...	b
-									
$A[1]$									
$A[2]$									
$\vdots$									
$A[i-1]$						x	x		
$A[i]$									
$\vdots$									
$A[n]$									

Lösung

Beispiel:

	0	1	2	3	4	5	6	7	8	9
-	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

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

↪ neue Laufzeit hängt von Grösse einer Zahl ab!
Laufzeit: $O(bn)$ ← Verhalten für $n \rightarrow \infty$, $b \rightarrow \infty$
Speicher: $O(bn)$ z.B. alle n Zahlen in A sind 64 bit aber b wird beliebig gross: braucht $\log_2(b)$ bits

⇒ Eingabegrösse ist nicht mehr n sondern $\Theta(n + \log(b))$

$S = 2^n$: Eingabe $\Theta(n)$, Laufzeit $\Theta(n 2^n)$ *exponentiell*

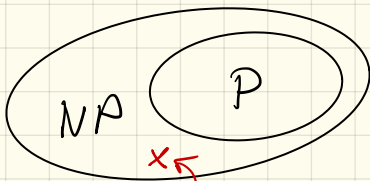
$S = n^c$: Eingabe $\Theta(n)$, Laufzeit $\Theta(n^{c+1})$ *polynomiell*

Man sagt: Laufzeit ist pseudopolynomiell

Subset sum ist ein Entscheidungsproblem:
Antwort ist ja oder nein

P : Menge aller Entscheidungsprobleme mit polynomieller Laufzeit

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



↪ subset sum

Vermutung: $P \neq NP$

Beweis ⇒ sehr schwierig (+1M USD)

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

Naiver Algorithmus: alle I ausprobieren,
bestes nehmen: Laufzeit $O(2^n)$

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

Kann definitiv schlecht sein!

Beispiel: $W = 10000g$ $(v_1, w_1) = (1CHF, 1g)$, $(v_2, w_2) = (9999, 10000)$

DP

Optimale Lösung enthält n oder nicht

$\Rightarrow$ optimale Lösung für n Gegenstände mit W
ist opt. Lösung für $n-1$ " mit W
oder " für $n-1$ " mit $W - w_n$

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

↑
Max value

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

Tabelle:

	0	1	2		$w-w_i$	w		w
0	0	0	0	...	...	...	...	0
1	0							
...	...							
$i-1$								
i								
...								
n	0							Lösung

keine Gegenstände

Lösung

Laufzeit/Speicher: $O(nW)$ (pseudopolynomial)

geht auch mit $O(nV)$, $V = v_1 + \dots + v_n$:

$H_{i,v}(i, v) = \min_{I \subseteq \{1..i\} \text{ Wert} \geq v \text{ zu erhalten}}$

$H_{i,v}(i, v) = \min(H_{i,v}(i-1, v), H_{i,v}(i-1, v-v_i) + v_i)$

	$v-v_i$	v		Lösung
$i-1$				
i				
n				Resultat: größtes Gewicht $\leq W$

monoton steigend

Wie können wir das beschleunigen?

Neue Idee: Berechnung einer approximativen Lösung

Input exakter: w_i, v_i, k ↗ $k \in \mathbb{N}$ INPUT
 Input approx: $w_i, \lfloor v_i/k \rfloor, k$ INPUT ← approximiert

Beispiel: Werte 112, 38, 1001, 17, 237, ...
 $k=10$, Werte 11, 7, 100, 1, 23, ...

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

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

$$O(n\bar{V}) \leq O(n^2 v_{\max} / k)$$

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

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

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

↗ es gilt $\geq$ aber wie weit voneinander entfernt

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

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

Es gilt: $\frac{v_i}{k} - 1 \leq \lfloor \frac{v_i}{k} \rfloor \leq \frac{v_i}{k} \Leftrightarrow v_i - k \leq k \lfloor \frac{v_i}{k} \rfloor \leq v_i$

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

$$\leq k \sum_{i \in \overline{\text{OPT}}} \lfloor \frac{v_i}{k} \rfloor$$

(Optimalität von $\overline{\text{OPT}}$ für Werte $\lfloor v_i/k \rfloor$)

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

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

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

$$\geq \text{Wert}(\text{OPT}) - n k$$

also: $\text{Wert}(\overline{\text{OPT}}) \geq \text{Wert}(\text{OPT}) - n k \geq (1 - \varepsilon) \text{Wert}(\text{OPT})$! $\leftarrow$ soll sein

herst.: $-n k \geq -\varepsilon \text{Wert}(\text{OPT})$

$$\Leftrightarrow k \leq \frac{\varepsilon}{n} \text{Wert}(\text{OPT})$$

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

$$\Rightarrow \text{Wert}(\text{OPT}) \geq v_{\max}$$

also wähle $k = \frac{\varepsilon}{n} v_{\max} \leq \frac{\varepsilon}{n} \text{Wert}(\text{OPT})$ erfüllt dann wie gewünscht

$$\Rightarrow \text{Laufzeit: } O(nV) \leq O(n^2 \frac{v_{\max}}{k})$$

$$= O(n^2 \frac{v_{\max}}{\frac{\varepsilon}{n} v_{\max}})$$

$$= O(n^3 / \varepsilon) \text{ polynom. in } n \text{ und } 1/\varepsilon$$

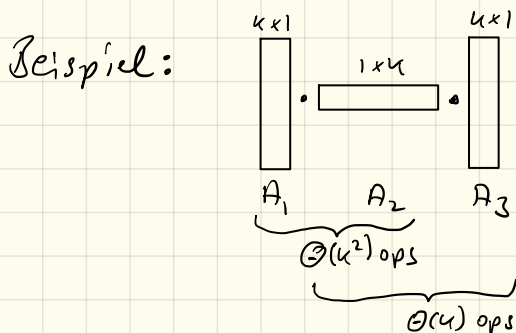
"Fully polynomial-time approximation scheme"

Matrix Kettenmultiplikation (optional)

Problem: Berechne $A_1 \cdot A_2 \cdot \dots \cdot A_n$ so günstig wie möglich. A_i sind Matrizen.

Freiheitsgrad: Assoziativität = Klammerung

z.B. $(A_1 A_2) A_3 = A_1 (A_2 A_3)$



$$(A_1 A_2) A_3 = \boxed{} \cdot \boxed{} = \boxed{} \quad \Theta(k^2) \text{ ops insgesamt}$$

$$A_1 (A_2 A_3) = \boxed{} \cdot \boxed{} = \boxed{} \quad \Theta(ku) \text{ ops insgesamt}$$

Idee: Betrachte die letzte Multiplikation einer optimalen Lösung

$$A_1 A_2 \dots A_n = \underbrace{(A_1 \dots A_i)}_{\text{optimal}} \underbrace{(A_{i+1} \dots A_n)}_{\text{optimal}}$$

Klammerung links/rechts ist optimal

$M(p, q) = \min \text{ Ops zur Berechnung}$
 Produkt $A_p \dots A_q$

Rekurrenz: Berechnung $O(q-p)$

$$M(p, q) = \min_{p \leq i < q} (M(p, i) + M(i+1, q) + \text{Kosten zur Berechnung } (A_p \dots A_i) \cdot (A_{i+1} \dots A_q))$$

Basis: $M(p, p) = 0, p = 1 \dots n$

In welcher Reihenfolge berechnen?
 Von kurzen zu langen Produkten, also von
 der Diagonale weg:

				q
	M			
		0		
			0	
				$\ddots$
				0
p				

* Lösung

In $M(p, q)$ gilt
 ja immer $p \leq q$

Laufzeit: $O(n^3)$ Speicher: $O(n^2)$

Beispiel A_1, A_2, A_3 von zuvor, betrachte nur Multi

	1	2	3
1	0	n^2	$2n$
2		0	n
3			0