Dynamic Programming II

Subset sum

- Input: $A[1 \dots n]$, b alle in $\mathbb{N}$
- Gesucht: $I \subseteq \{1, \dots, n\}$, $\sum_{i \in I} A[i] = b$

Naiv: alle Teilsummen: 2^n

Rekursion: zwei Möglichkeiten (i) benutze $A[i]$
(ii) benutze $A[i]$ nicht

$T(i,s)$ = " s ist Teilsumme von $A[1 \dots i]$

$\Rightarrow T(i,s) = T(i-1, s) \vee T(i-1, s - A[i])$

$T(i-1, s-A[i])$: 0, falls $s - A[i] < 0$

Base cases: $T(i,0) = 1$, $T(0,s) = 0$ $s \geq 1$

Beispiel: $A = [5, 3, 10, 3, 1]$

$T(i,s)$	0	1	2	3	4	5	6	7 $= b$
$i=0$, –	1	0	0	0	0	0	0	0
1, 5	1	0	0	0	0	1	0	0
2, 3	1	0	0	1	0	1	0	0
3, 10	1	0	0	1	0	1	0	0
4, 3	1	0	0	1	0	1	1	0
5, 1	1	1	0	1	1	1	1	1

$\Theta(n \cdot b)$

Knapsack

- **Input:**
  - Rucksack mit Gewichtslimit W
  - n items (i, w_i, p_i), w_i weight, p_i profit $\quad i \in [n]$
- **Gesucht:** $I \subseteq \{1, \dots, n\}$, $\sum_{i \in I} p_i$ maximal mit $\sum_{i \in I} w_i \leq W$.

Naiv: alle Teilsummen : 2^n $\quad \mathcal{P}(\{1, \dots, n\})$

Rekursion: $G(i, v)$ ist minimales Gewicht, um mit ersten i items profit v zu erreichen.

zwei Möglichkeiten: (i) item i ist nicht im Rucksack
(ii) item i kommt in den Rucksack

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

$G(i-1, v-v_i)$: 0, falls $v - v_i \leq 0$

Base cases: $G(0,0) = 0, \quad G(0, v) = \infty$

Beispiel: (i, w_i, p_i): $(1,1,1)$, $(2,2,4)$, $(3,3,5)$

$= P = \sum_i p_i$

$G(i,s)$	0	1	2	3	4	5	6	7	8	9	10
$i=0$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1, (1,1)	0	1	∞	∞	∞	∞	∞	∞	∞	∞	∞
2, (2,4)	0	1	∞	∞	2	3	∞	∞	∞	∞	∞
3, (3,5)	0	1	∞	∞	2	3	4	∞	∞	5	6

Approximation Knapsack

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

0, falls $v - v_i < 0$

Base cases: $G(0,0) = 0, \quad G(0,v) = \infty$

Beispiel: (i, w_i, p_i): $(1,1,1)$, $(2,2,4)$, $(3,3,5)$

$$p_i' = K \cdot \left\lfloor \frac{p_i}{K} \right\rfloor,$$

$K=2 \Rightarrow (i, w_i, p_i')$: $(1,1,0)$, $(2,2,4)$, $(3,3,4)$

$= P = \sum_i p_i$

$G(i,s)$	0	1	2	3	4	5	6	7	8	9	10
$i=0$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1, (1,1)	0	1	∞	∞	∞	∞	∞	∞	∞	∞	∞
2, (2,4)	0	1	∞	∞	2	3	∞	∞	∞	∞	∞
3, (3,5)	0	1	∞	∞	2	3	4	∞	∞	5	6

$G(i,s)$	0	2	4	6	8
$i=0$	0	∞	∞	∞	∞
1, (1,0)	0	∞	∞	∞	∞
2, (2,4)	0	∞	2	∞	∞
3, (3,4)	0	∞	2	∞	5

Längste aufsteigende Teilfolge

- Input: $A[1 \dots n]$ aus $\mathbb{Z}$
- Gesucht: Länge einer längsten aufsteigenden Teilfolge

2 13 17 9 11

Idee: 1 5 2 6 3 ...

nehme 1, 2, 3 so dass Ende am kleinsten.

Rekursion: $M(i, \ell)$ = kleinste Endung einer Teilfolge der Länge ℓ von $A[1..i]$

$$M(1, \ell) = \begin{cases} A[1] & , \ \ell = 1 \\ \infty & \text{sonst} \end{cases}$$

$i \geq 2$: (i) verwende $A[i]$ nicht.

(ii) verwende $A[i]$. (nur wenn $M(i-1, \ell-1) < A[i]$.)

$$M(i, \ell) = \begin{cases} A[1] & i = \ell = 1 \\ \infty & i = 1, \ \ell > 1 \\ \min\{M(i-1, \ell), A[i]\} & i \geq 2, \ M(i-1, \ell-1) < A[i] \\ M(i-1, \ell) & \text{sonst} \end{cases}$$

Beispiel $A[1..5] = [3, 7, 8, 4, 5]$

$M(i,\ell)$	1	2	3	4	5
$i=1,\ 3$	3	∞	∞	∞	∞
$2,\ 7$	3	7	∞	∞	∞
$3,\ 8$	3	7	8	∞	∞
$4,\ 4$	3	4	8	∞	∞
$5,\ 5$	3	4	5	∞	∞

Idee: $(3, 4) \longrightarrow (3, 4, 5)$
$\ell \qquad\qquad\qquad \ell+1$

erkenne: $M(i, \ell) \leq M(i, \ell+1)$

also: Zeilen sind sortiert!

Idee: suche Stelle, die wir verbessern können

$A[1..5] = [3, 7, 8, 4, 99, 6, 100, 7, 101, 8]$

	1	2	3	4	5	6	7	8	9	10
$M(*,\ell)$	3	~~7~~	~~8~~	~~99~~	~~100~~	~~101~~	∞	∞	∞	∞
		4	5	6	7	8				

Algorithms and Data Structures Week 8

Georg Hasebe, 4th October.

Längste Aufsteigende Teilfolge

Mit der LAT (längste aufsteigende Teilfolge) sind wir leider nicht mehr ganz fertig geworden, daher hier nochmals ein kurzer Überblick: Neben der Version mit dem $O(n^2)$ Table gibt es auch die Version mit dem $O(n)$ Array. Die Idee ist dabei wie folgt:

Angenommen, wir haben das Array $[3, 4, 99, 5, 100, 6, 1, 2]$. Der Verlauf für die ersten drei Elemente ist eindeutig, denn $3, 4, 99$ ist ja eine LAT.

Nun bekommen wir die 4 und es ist klar, dass wir damit unsere bisherige LAT nicht erweitern können. Was nun? Wir erinnern uns daran, dass wir stets versucht haben, die LAT mit der kleinsten Endung von einer bestimmten Länge zu speichern. Die LAT $3, 4, 99$ könnten wir uns also auch so vorstellen:

3
$3, 4$
$3, 4, 99$

Die erste Zeile stellt eine LAT der Länge 1 mit der kleinsten Endung dar, die zweite Zeile die LAT der Länge 2 mit kleinster Endung, und so weiter.

Nun stellen wir uns die Frage, welche dieser LATs das neue Element 5 verbessern kann. Antwort: nur $3, 4, 99$, da dies die einzige LAT ist, bei der wir die Endung verkleinern könnten. Die bisher beste LAT (mit der kleinsten Endung) wäre somit $3, 4, 5$.

Als Nächstes haben wir 100. Hier ist es einfach: Wir erweitern unsere LAT zu $3, 4, 5, 100$. Dann bekommen wir die 6 und stellen uns dieselbe Frage wie zuvor:

3
$3, 4$
$3, 4, 5$
$3, 4, 5, 100$

Es ist leicht zu erkennen, dass die 6 nur eine Verbesserung für die LAT $3, 4, 5, 100$ darstellt, die somit zu $3, 4, 5, 6$ wird. Nun bekommen wir die 1 und stellen uns dieselbe Frage wie zuvor:

3
$3, 4$

$3, 4, 5$
$3, 4, 5, 6$

Wir stellen fest, dass dieser Fall etwas anders ist als bisher, denn $3, 4, 5, 1$ wäre nicht mehr aufsteigend. Was ist aber mit der LAT der Länge 1, nämlich 3? Wenn die 3 durch die 1 ersetzt wird, hätten wir eine LAT der Länge 1 mit der kleinsten Endung 1, was eine Verbesserung darstellt. Wir halten also fest:

1
$3, 4$
$3, 4, 5$
$3, 4, 5, 6$

Zu guter Letzt bekommen wir die 2. Auch hier ist die Situation wieder etwas anders. Wir könnten beispielsweise nicht einfach $3, 4$ zu $2, 4$ ändern, da 4 viel früher im Array kam und $2, 4$ keine LAT der Länge 2 in unserem Array wäre. Allerdings könnten wir die 2 an die LAT 1 der Länge 1 anhängen, womit wir $1, 2$ erhalten, was eine Verbesserung zu $3, 4$ darstellt. Wir halten also wieder fest:

1
$1, 2$
$3, 4, 5$
$3, 4, 5, 6$

Aber eigentlich wollten wir doch nur ein einzelnes Array der Länge n, nicht mehrere Arrays, die LATs verschiedener Längen speichern. Überlegen wir uns anhand der oben angegebenen Tabellen, was wir tatsächlich gemacht haben: Wir haben immer nach einem Element gesucht, das kleiner oder gleich dem neuen Element war, und dieses dann durch das neue ersetzt. Tatsächlich können wir dies auch tun, wenn wir nur ein einzelnes Array haben:

3
$3, 4$
$3, 4, 99$

wird zu

$3, 4, 99$

und damit wird (Platz für 100 finden)

3
$3, 4$
$3, 4, 5$
$3, 4, 5, 100$

zu

$3, 4, 5, 100$

und so weiter. Mit anderen Worten, der Verlauf bei unserem Beispiel sieht etwa so aus:

$$3 \to 3, 4 \to 3, 4, 99 \to 3, 4, 5 \to 3, 4, 5, 100 \to 3, 4, 5, 6 \to 1, 4, 5, 6 \to 1, 2, 5, 6$$

Dabei haben wir im vorletzten Schritt die 3 zu einer 1 verbessert und im letzten Schritt die 4 zu einer 2 verbessert. In all diesen Schritten arbeiten wir nur mit einem einzelnen Array, das gewissermaßen als komprimierte Version der vielen Arrays von oben angesehen werden kann. Wenn wir ein Array der Länge n haben, initialisiert mit ∞ an allen Stellen, dann würde unser Array am Ende so aussehen:

$$[1, 2, 4, 5, \infty, \infty, \infty]$$

Die Antwort (also die Länge einer LAT) wäre somit einfach der größte Index, bei dem kein ∞ steht, hier also 4. Bemerke, dass $1, 2, 4, 5$ keine LAT in unserem Array wäre, eine Methode zur Rekonstruktion wird im Skript beschrieben.