

21. Dynamic Programming III

FPTAS [Ottman/Widmayer, Kap. 7.2, 7.3, Cormen et al, Kap. 15,35.5]

Approximation

Sei ein $\varepsilon \in (0, 1)$ gegeben. Sei I_{opt} eine bestmögliche Auswahl.
Suchen eine gültige Auswahl I mit

$$\sum_{i \in I} v_i \geq (1 - \varepsilon) \sum_{i \in I_{\text{opt}}} v_i.$$

Summe der Gewichte darf W natürlich in keinem Fall überschreiten.

575

576

Andere Formulierung des Algorithmus

Bisher: Gewichtsschranke $w \rightarrow$ maximaler Nutzen v

Umkehrung Nutzen $v \rightarrow$ minimales Gewicht w

\Rightarrow **Alternative Tabelle:** $g[i, v]$ gibt das minimale Gewicht an, welches

- eine Auswahl der ersten i Gegenstände ($0 \leq i \leq n$) hat, die
- einen Nutzen von genau v aufweist ($0 \leq v \leq \sum_{i=1}^n v_i$).

Berechnung

Initial

- $g[0, 0] \leftarrow 0$
- $g[0, v] \leftarrow \infty$ (Nutzen v kann mit 0 Gegenständen nie erreicht werden.).

Berechnung

$$g[i, v] \leftarrow \begin{cases} g[i-1, v] & \text{falls } v < v_i \\ \min\{g[i-1, v], g[i-1, v-v_i] + w_i\} & \text{sonst.} \end{cases}$$

aufsteigend nach i und für festes i aufsteigend nach v .

Lösung ist der grösste Index v mit $g[n, v] \leq w$.

577

578

Beispiel

$$E = \{(2, 3), (4, 5), (1, 1)\}$$

	v									
	0	1	2	3	4	5	6	7	8	9
\emptyset	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
(2, 3)	0	∞	∞	2	∞	∞	∞	∞	∞	∞
(4, 5)	0	∞	∞	2	∞	4	∞	∞	6	∞
(1, 1)	0	1	∞	2	3	4	5	∞	6	7

Auslesen der Lösung: wenn $g[i, v] = g[i - 1, v]$ dann Gegenstand i nicht benutzt und bei $g[i - 1, v]$ weiterfahren, andernfalls benutzt und bei $g[i - 1, v - v_i]$ weiterfahren.

579

580

Der Approximationstrick

Pseudopolynomielle Laufzeit wird polynomiell, wenn vorkommenden Werte in Polynom der Eingabelänge beschränkt werden können.

Sei $K > 0$ *geeignet* gewählt. Ersetze die Nutzwerte v_i durch "gerundete Werte" $\tilde{v}_i = \lfloor v_i/K \rfloor$ und erhalte eine neue Eingabe $E' = (w_i, \tilde{v}_i)_{i=1 \dots n}$.

Wenden nun den Algorithmus auf Eingabe E' mit derselben Gewichtsschranke W an.

Idee

Beispiel $K = 5$

Eingabe Nutzwerte

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ..., 98, 99, 100

→

0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, ..., 19, 19, 20

Offensichtlich weniger unterschiedliche Nutzwerte

Eigenschaften des neuen Algorithmus

- Auswahl von Gegenständen aus E' ist genauso gültig wie die aus E . Gewicht unverändert!
- Laufzeit des Algorithmus ist beschränkt durch $\mathcal{O}(n^2 \cdot v_{\max}/K)$
($v_{\max} := \max\{v_i | 1 \leq i \leq n\}$)

581

582

Wie gut ist die Approximation?

Es gilt

$$v_i - K \leq K \cdot \left\lfloor \frac{v_i}{K} \right\rfloor = K \cdot \tilde{v}_i \leq v_i$$

Sei I'_{opt} eine optimale Lösung von E' . Damit

$$\begin{aligned} \left(\sum_{i \in I_{opt}} v_i \right) - n \cdot K &\stackrel{|I_{opt}| \leq n}{\leq} \sum_{i \in I_{opt}} (v_i - K) \leq \sum_{i \in I_{opt}} (K \cdot \tilde{v}_i) = K \sum_{i \in I_{opt}} \tilde{v}_i \\ &\stackrel{I'_{opt} \text{ optimal}}{\leq} K \sum_{i \in I'_{opt}} \tilde{v}_i = \sum_{i \in I'_{opt}} K \cdot \tilde{v}_i \leq \sum_{i \in I'_{opt}} v_i. \end{aligned}$$

583

Wahl von K

Forderung:

$$\sum_{i \in I'} v_i \geq (1 - \varepsilon) \sum_{i \in I_{opt}} v_i.$$

Ungleichung von oben:

$$\sum_{i \in I'_{opt}} v_i \geq \left(\sum_{i \in I_{opt}} v_i \right) - n \cdot K$$

$$\text{Also: } K = \varepsilon \frac{\sum_{i \in I_{opt}} v_i}{n}.$$

584

Wahl von K

Wähle $K = \varepsilon \frac{\sum_{i \in I_{opt}} v_i}{n}$. Die optimale Summe ist aber unbekannt, daher wählen wir $K' = \varepsilon \frac{v_{\max}}{n}$.³⁵

Es gilt $v_{\max} \leq \sum_{i \in I_{opt}} v_i$ und somit $K' \leq K$ und die Approximation ist sogar etwas besser.

Die Laufzeit des Algorithmus ist beschränkt durch

$$\mathcal{O}(n^2 \cdot v_{\max}/K') = \mathcal{O}(n^2 \cdot v_{\max}/(\varepsilon \cdot v_{\max}/n)) = \mathcal{O}(n^3/\varepsilon).$$

³⁵Wir können annehmen, dass vorgängig alle Gegenstände i mit $w_i > W$ entfernt wurden.

585

FPTAS

Solche Familie von Algorithmen nennt man **Approximationsschema**: die Wahl von ε steuert Laufzeit und Approximationsgüte.

Die Laufzeit $\mathcal{O}(n^3/\varepsilon)$ ist ein Polynom in n und in $\frac{1}{\varepsilon}$. Daher nennt man das Verfahren auch ein voll polynomiell Approximationsschema

FPTAS - Fully Polynomial Time Approximation Scheme

586

22. Gierige (Greedy) Algorithmen

Gebrochenes Rucksack Problem, Huffman Coding [Cormen et al, Kap. 16.1, 16.3]

Das Gebrochene Rucksackproblem

Menge von $n \in \mathbb{N}$ Gegenständen $\{1, \dots, n\}$ gegeben. Jeder Gegenstand i hat Nutzwert $v_i \in \mathbb{N}$ und Gewicht $w_i \in \mathbb{N}$. Das Maximalgewicht ist gegeben als $W \in \mathbb{N}$. Bezeichnen die Eingabe mit $E = (v_i, w_i)_{i=1, \dots, n}$.

Gesucht: Anteile $0 \leq q_i \leq 1$ ($1 \leq i \leq n$) die die Summe $\sum_{i=1}^n q_i \cdot v_i$ maximieren unter $\sum_{i=1}^n q_i \cdot w_i \leq W$.

587

588

Gierige (Greedy) Heuristik

Sortiere die Gegenstände absteigend nach Nutzen pro Gewicht v_i/w_i .

Annahme $v_i/w_i \geq v_{i+1}/w_{i+1}$

Sei $j = \max\{0 \leq k \leq n : \sum_{i=1}^k w_i \leq W\}$. Setze

- $q_i = 1$ für alle $1 \leq i \leq j$.
- $q_{j+1} = \frac{W - \sum_{i=1}^j w_i}{w_{j+1}}$.
- $q_i = 0$ für alle $i > j + 1$.

Das ist schnell: $\Theta(n \log n)$ für Sortieren und $\Theta(n)$ für die Berechnung der q_i .

589

Korrektheit

Annahme: Optimale Lösung (r_i) ($1 \leq i \leq n$).

Der Rucksack wird immer ganz gefüllt: $\sum_i r_i \cdot w_i = \sum_i q_i \cdot w_i = W$.

Betrachte k : kleinstes i mit $r_i \neq q_i$. Die gierige Heuristik nimmt per Definition so viel wie möglich: $q_k > r_k$. Sei $x = q_k - r_k > 0$.

Konstruiere eine neue Lösung (r'_i) : $r'_i = r_i \forall i < k$. $r'_k = q_k$. Entferne Gewicht $\sum_{i=k+1}^n \delta_i = x \cdot w_k$ von den Gegenständen $k + 1$ bis n . Das geht, denn $\sum_{i=k}^n r_i \cdot w_i = \sum_{i=k}^n q_i \cdot w_i$.

590

Korrektheit

$$\begin{aligned} \sum_{i=k}^n r'_i v_i &= r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^n (r_i w_i - \delta_i) \frac{v_i}{w_i} \\ &\geq r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^n r_i w_i \frac{v_i}{w_i} - \delta_i \frac{v_k}{w_k} \\ &= r_k v_k + x w_k \frac{v_k}{w_k} - x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^n r_i w_i \frac{v_i}{w_i} = \sum_{i=k}^n r_i v_i. \end{aligned}$$

Also ist (r'_i) auch optimal. Iterative Anwendung dieser Idee erzeugt die Lösung (q_i) .

591

Huffman-Codierungen

Ziel: Speicherplatzeffizientes Speichern einer Folge von Zeichen mit einem binären *Zeichencode* aus *Codewörtern*.

Beispiel

File aus 100.000 Buchstaben aus dem Alphabet $\{a, \dots, f\}$

	a	b	c	d	e	f
Häufigkeit (Tausend)	45	13	12	16	9	5
Codewort fester Länge	000	001	010	011	100	101
Codewort variabler Länge	0	101	100	111	1101	1100

Speichergröße (Code fixe Länge): 300.000 bits.

Speichergröße (Code variabler Länge): 224.000 bits.

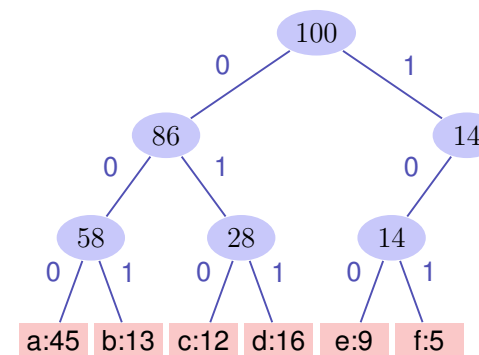
592

Huffman-Codierungen

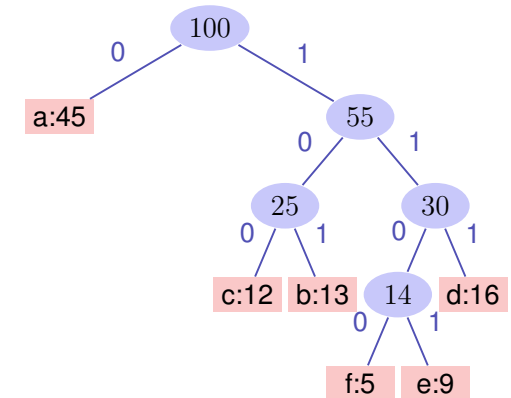
- Betrachten *Präfixcodes*: kein Codewort kann mit einem anderen Codewort beginnen.
- Präfixcodes können im Vergleich mit allen Codes die optimale *Datenkompression* erreichen (hier ohne Beweis).
- Codierung: Verkettung der Codewörter ohne Zwischenzeichen (Unterschied zum Morsen!)
 $af fe \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow 0110011001101$
- Decodierung einfach da Präfixcode
 $0110011001101 \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow af fe$

593

Codebäume



Codewörter fixer Länge



Codewörter variabler Länge

594

Eigenschaften der Codebäume

- Optimale Codierung eines Files wird immer durch vollständigen binären Baum dargestellt: jeder innere Knoten hat zwei Kinder.
- Sei C die Menge der Codewörter, $f(c)$ die Häufigkeit eines Codeworts c und $d_T(c)$ die Tiefe eines Wortes im Baum T . Definieren die **Kosten** eines Baumes als

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c).$$

(Kosten = Anzahl Bits des codierten Files)

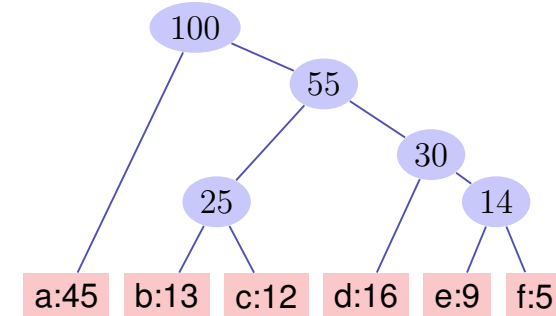
Bezeichnen im folgenden einen Codebaum als optimal, wenn er die Kosten minimiert.

595

Algorithmus Idee

Baum Konstruktion von unten nach oben

- Starte mit der Menge C der Codewörter
- Ersetze iterativ die beiden Knoten mit kleinster Häufigkeit durch ihren neuen Vaterknoten.



596

Algorithmus Huffman(C)

Input : Codewörter $c \in C$
Output : Wurzel eines optimalen Codebaums

```
n ← |C|
Q ← C
for i = 1 to n - 1 do
    Alloziere neuen Knoten z
    z.left ← ExtractMin(Q)           // Extrahiere Wort mit minimaler Häufigkeit.
    z.right ← ExtractMin(Q)
    z.freq ← z.left.freq + z.right.freq
    Insert(Q, z)
return ExtractMin(Q)
```

597

Analyse

Verwendung eines Heaps: Heap bauen in $\mathcal{O}(n)$. Extract-Min in $\mathcal{O}(\log n)$ für n Elemente. Somit Laufzeit $\mathcal{O}(n \log n)$.

598

Das gierige Verfahren ist korrekt

Theorem

Seien x, y zwei Symbole mit kleinsten Frequenzen in C und sei $T'(C')$ der optimale Baum zum Alphabet $C' = C - \{x, y\} + \{z\}$ mit neuem Symbol z mit $f(z) = f(x) + f(y)$. Dann ist der Baum $T(C)$ der aus $T'(C')$ entsteht, indem der Knoten z durch einen inneren Knoten mit Kindern x und y ersetzt wird, ein optimaler Codebaum zum Alphabet C .

Beweis

Es gilt $f(x) \cdot d_T(x) + f(y) \cdot d_T(y) = (f(x) + f(y)) \cdot (d_{T'}(z) + 1) = f(z) \cdot d_{T'}(z) + f(x) + f(y)$. Also $B(T') = B(T) - f(x) - f(y)$.

Annahme: T sei nicht optimal. Dann existiert ein optimaler Baum T'' mit $B(T'') < B(T)$. Annahme: x und y Brüder in T'' . T''' sei der Baum T'' in dem der innere Knoten mit Kindern x und y gegen z getauscht wird. Dann gilt

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T').$$

Widerspruch zur Optimalität von T' .

Die Annahme, dass x und y Brüder sind in T'' kann man rechtfertigen, da ein Tausch der Elemente mit kleinster Häufigkeit auf die unterste Ebene den Wert von B höchstens verkleinern kann.