Zwei Beispiele zum dynamischen Programmieren

Wolfgang Mulzer

1 Einkaufsproblem (auch Rucksackproblem genannt)

gegeben: Eine Menge von n Artikeln, mit positiven ganzzahligen Preisen $p_1, \ldots, p_n \in \mathbb{N}$ und positiven ganzzahligen Werten $w_1, \ldots, w_n \in \mathbb{N}$. Ein Budget $B \in \mathbb{N}$.

gesucht: Eine Teilmenge von Artikeln $A \subseteq \{1, \ldots, n\}$, so dass der Gesamtpreis $p_A \leq B$ ist und der Gesamtwert w_A maximal ist.

Lösung mit dynamischem Programmieren:

1. Finde geeignete Teilprobleme: Sei $\mathtt{E}[i, b]$ der Wert eines optimalen Einkaufs, wenn nur die Artikel $1, \ldots, i$ erlaubt sind und unser Budget auf b beschränkt ist.

2. Finde eine Rekursion für $\mathtt{E}[i, b]$:

   Wenn wir kein Geld haben, können wir nichts kaufen:

   $$\mathtt{E}[i, 0] = 0, \qquad \text{für } i = 0, \ldots, n.$$

   Wenn wir nichts kaufen können, können wir keinen Wert erzielen:

   $$\mathtt{E}[0, b] = 0, \qquad \text{für } b = 0, \ldots, B.$$

   Wenn wir den i-ten Artikel kaufen, erzielen wir einen Wert von w_i, haben aber nur noch $b - p_i$ Geld zur Verfügung. Wenn wir den i-ten Artikel nicht kaufen, erzielen wir keinen Wert, aber haben immer noch b Geld übrig.

   $$\mathtt{E}[i, b] = \begin{cases} \mathtt{E}[i-1, b], & \text{falls } p_i > b, \\ \max\{\mathtt{E}[i-1, b], w_i + \mathtt{E}[i-1, b-p_i]\}, & \text{sonst.} \end{cases}$$

3. Implementiere die Rekursion:

```
for i := 0 to n do
    E[i, 0] <- 0
for b := 0 to B do
    E[0, b] <- 0
for i := 1 to n do
    for b := 1 to B do
        if p(i) > b then
            E[i, b] <- E[i - 1, b]
        else
            E[i, b] <- max{E[i - 1, b], w(i) + E[i - 1, b - p(i)]}
return E[n, B]
```

4. Finde einen optimalen Einkauf:

   Möglichkeit 1: Führe eine zusätzliche Tabelle `kaufen`$[i, b]$ ein. Dabei bedeutet `kaufen`$[i, b]$, dass wir den Artikel i kaufen, wenn wir eine Budget von b zur Verfügung haben. Fülle `kaufen`$[i, b]$ zusammen mit `E`$[i, b]$ aus und benutze `kaufen`$[i, b]$, um hinterher den Einkauf zu bestimmen.

```
for i := 0 to n do
    E[i, 0] <- 0
for b := 0 to B do
    E[0, b] <- 0
for i := 1 to n do
    for b := 1 to B do
        if p(i) > b then
            E[i, b] <- E[i - 1, b]
            kaufen[i, b] <- false  // ***
        else if E[i - 1, b] >= w(i) + E[i - 1, b - p(i)] then
            E[i, b] <- E[i - 1, b]
            kaufen[i, b] <- false  // ***
        else
            E[i, b] <- w(i) + E[i - 1, b - p(i)]
            kaufen[i, b] <- true  // ***
// Ausgabe eines optimalen Einkaufs
b <- B
for i := n downto 1 do
    if kaufen[i, b] then
        print "Kaufe i"
        b <- b - p(i)
```

   Möglichkeit 2: Benutze die Tabelle `E`$[i, b]$ direkt, um einen optimalen Einkauf zu rekonstruieren, indem du die Einträge hernimmst, um festzustellen, woher das Maximum kommt.

```
// die Tabelle E[i, b] ist mit dem obigen Algorithmus berechnet worden
b <- B
for i := n downto 1 do
    // der Eintrag von E[i, b] stammt entweder von E[i - 1, b] oder von
    // w(i) + E[i-1, b - p(i)].
    // Nur in letzterem Fall wollen wir i kaufen.
    if E[i, b] != E[i - 1, b] then
        print "Kaufe i"
        b <- b - p(i)
```

Die Laufzeit und der Speicherbedarf des Algorithmus sind $O(nB)$. Dies ist nur gut, wenn B klein ist. Im allgemeinen ist die Laufzeit nicht polynomiell in der Eingabelänge, da zur Kodierung von B nur $O(\log B)$ Bits nötig sind. Daher sagt man, dass der obige Algorithmus *pseudo-polynomielle Laufzeit* hat. Das Einkaufsproblem ist *schwach* NP-*vollständig*, also gibt es wahrscheinlich keinen wesentlich besseren Algorithmus.

2 Rundreiseproblem (Traveling-Salesperson Problem, TSP)

gegeben: n Städte $0, 1, \ldots, n-1$ mit paarweisen Abständen $d(i, j) = d(j, i) > 0$.

gesucht: Eine Reihenfolge π, welche die Gesamtlänge $\sum_{i=0}^{n-1} d(\pi(i), \pi(i+1))$ minimiert. Dabei beginnen wir mit Stadt 0 und enden in Stadt 0 (das Argument von π wird also modulo n gerechnet).

Naive Lösung: Probiere alle möglichen Besuchsreihenfolgen π durch. Davon gibt es $(n-1)!$ Stück, da wir in Stadt 0 anfangen und aufhören. Für jede Reihenfolge benötigen wir lineare Zeit, um die Gesamtlänge zu berechnen. Wir erhalten also die Laufzeit $O(n!)$, was für $n > 10$ hoffnungslos ist.

Lösung mit dynamischem Programmieren:

1. Finde geeignete Teilprobleme: Sei S eine Teilmenge von $\{1, \ldots, n-1\}$ und $m \in \{0, \ldots, n-1\} \setminus S$ eine Stadt. Wir definieren $\mathtt{T}[S, m]$ als die Länge einer optimalen Tour, die in Stadt 0 anfängt, in Stadt m aufhört, und dazwischen genau die Städte aus S genau einmal besucht.

2. Finde eine Rekursion für $\mathtt{T}[S, m]$:

   Wenn wir zwischen 0 und m keine Städte besuchen, so ist die Länge der optimalen Tour von 0 nach m genau $d(0, m)$.

   $$\mathtt{T}[\emptyset, m] = d(0, m), \qquad \text{für } m = 0, \ldots, n-1.$$

   Ansonsten unterscheiden wir, welche der Städte aus S, die wir auf der Tour von 0 nach m besuchen, die letzte ist. Wir nennen diese Stadt a. Dann ist die Teil-Tour von 0 nach a auch optimal, und sie besucht vorher genau die Städte aus $S \setminus \{a\}$.

   $$\mathtt{T}[S, m] = \max_{a \in S} \{\mathtt{T}[S \setminus \{a\}, a] + d(a, m)\}.$$

3. Implementiere die Rekursion: Übung

4. Finde eine optimale Tour: Übung

Man erhält eine Laufzeit von $O(n^2 2^n)$ und Platzbedarf $O(2^n)$.