Dynamische Programmierung – Aufbau aus Bausteinen

Anwendbar, wenn das **Optimalitätsprinzip** gilt:

- Optimale Lösungen bestehen aus optimalen Löungen für Teilprobleme.
- Mehrere optimale Lösungen $\Rightarrow$ es is egal, welche benutzt wird.

Beispiel: Rucksackproblem

Annahme: ganzzahlige Gewichte
$P(i, C)$:= optimaler Profit für Gegenstände $1,\ldots,i$
unter Benutzung von Kapazität $\leq C$.

$P(0, C) := 0$

Lemma:

$$\forall 1 \leq i \leq n : P(i, C) = \max(P(i-1, C), P(i-1, C - w_i) + p_i)$$

Dynamische Programmierung

auszufüllende Tabelle

Wdh. Lemma:

$$\forall 1 \leq i \leq n : P(i, C) = \max(P(i-1, C), P(i-1, C-w_i) + p_i)$$

i / C	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Beweis des Lemmas

$P(i, C)$:= optimaler Profit für Gegenstände $1,\ldots,i$ bei Kap. C.

Lemma: $P(i,C) = \max(P(i-1,C), P(i-1,C-w_i)+p_i)$

Beweis:

Sei $\mathbf{x}$ optimale Lösung für Objekte $1..i$, Kapazität C, d. h.
$\mathbf{c}\cdot\mathbf{x} = P(i,C)$.

Beweis des Lemmas

$P(i, C)$:= optimaler Profit für Gegenstände $1,\ldots,i$ bei Kap. C.

Lemma: $P(i,C) = \max(P(i-1,C), P(i-1,C-w_i)+p_i)$

Beweis:

Sei $\mathbf{x}$ optimale Lösung für Objekte $1..i$, Kapazität C, d. h.
$\mathbf{c}\cdot\mathbf{x} = P(i,C)$.

Fall $x_i = 0$:

$\Rightarrow \mathbf{x}$ ist auch (opt.) Lösung für Objekte $1..i-1$, Kapazität C.

$\Rightarrow P(i,C) = \mathbf{c}\cdot\mathbf{x} = P(i-1,C)$

Beweis des Lemmas

$P(i,C)$:= optimaler Profit für Gegenstände $1,\ldots,i$ bei Kap. C.

Lemma: $P(i,C) = \max(P(i-1,C), P(i-1,C-w_i)+p_i)$

Beweis:

Sei $\mathbf{x}$ optimale Lösung für Objekte $1..i$, Kapazität C, d. h.
$\mathbf{c}\cdot\mathbf{x} = P(i,C)$.

Fall $x_i = 0$:

$\Rightarrow \mathbf{x}$ ist auch (opt.) Lösung für Objekte $1..i-1$, Kapazität C.

$\Rightarrow P(i,C) = \mathbf{c}\cdot\mathbf{x} = P(i-1,C)$

Fall $x_i = 1$:

$\Rightarrow \mathbf{x}$ ohne i ist auch Lösung für Objekte $1..i-1$, Kapazität $C-w_i$.

Wegen Austauschbarkeit muß $\mathbf{x}$ ohne i optimal für diesen Fall sein.

$\Rightarrow P(i,C) - p_i = P(i-1,C-w_i)$

$\Leftrightarrow P(i,C) = P(i-1,C-w_i)+p_i$

Beweis des Lemmas

$P(i,C)$:= optimaler Profit für Gegenstände $1,\ldots,i$ bei Kap. C.

Lemma: $P(i,C) = \max(P(i-1,C), P(i-1,C-w_i)+p_i)$

Beweis:

Sei $\mathbf{x}$ optimale Lösung für Objekte $1..i$, Kapazität C, d. h.
$\mathbf{c}\cdot\mathbf{x} = P(i,C)$.

Fall $x_i = 0$:

$\Rightarrow \mathbf{x}$ ist auch (opt.) Lösung für Objekte $1..i-1$, Kapazität C.

$\Rightarrow P(i,C) = \mathbf{c}\cdot\mathbf{x} = P(i-1,C)$

Fall $x_i = 1$:

$\Rightarrow \mathbf{x}$ ohne i ist auch Lösung für Objekte $1..i-1$, Kapazität $C-w_i$.

Wegen Austauschbarkeit muß $\mathbf{x}$ ohne i optimal für diesen Fall sein.

$\Rightarrow P(i,C) - p_i = P(i-1,C-w_i)$

$\Leftrightarrow P(i,C) = P(i-1,C-w_i)+p_i$

Insgesamt, wegen Optimalität von $\mathbf{x}$,

$P(i,C) = \max(P(i-1,C), P(i-1,C-w_i)+p_i)$ □

Berechung von $P(i, C)$ elementweise:

$$P(i,C) = \max(P(i-1,C), P(i-1, C-w_i)+p_i)$$

i / C	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Procedure knapsack($\mathbf{p}$, $\mathbf{c}$, n, M)
  array P$[0 \ldots M] = [0, \ldots, 0]$
  bitarray decision$[1 \ldots n, 0 \ldots M] = [(0, \ldots, 0), \ldots, (0, \ldots, 0)]$
  for $i := 1$ **to** n **do**
    // invariant: $\forall C \in \{1, \ldots, M\} : \mathrm{P}[C] = P(i-1, C)$
    for $C := M$ **downto** w_i **do**
      if $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$ **then**
        $\mathrm{P}[C] := \mathrm{P}[C - w_i] + p_i$
        decision$[i, C] := 1$

Rekonstruktion der Lösung

$C := M$
array $\mathsf{x}[1 \ldots n]$
for $i := n$ **downto** 1 **do**
    $\mathsf{x}[i] :=$ decision$[i, C]$
    if $\mathsf{x}[i] = 1$ **then** $C := C - w_i$
endfor
return x

Analyse:

Zeit: $\mathrm{O}(nM)$ pseudopolynomiell

Platz: $M + \mathrm{O}(n)$ Maschinenwörter plus nM bits.

Beispiel

Maximiere $(10, 20, 15, 20) \cdot \mathbf{x}$,
so dass $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2						
3						
4						

Beispiel

Maximiere $(10, 20, 15, 20) \cdot \mathbf{x}$,
so dass $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3						
4						

Beispiel

Maximiere $(10, 20, 15, 20) \cdot \mathbf{x}$,
so dass $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4						

Beispiel

Maximiere $(10, 20, 15, 20) \cdot \mathbf{x}$,
so dass $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4	0, (0)	10, (0)	15, (0)	25, (0)	30, (0)	35, (0)

Algorithmenentwurf mittels dynamischer Programmierung

1. Was sind die Teilprobleme? — Kreativität!
2. Wie setzen sich optimale Lösungen aus Teilproblemlösungen zusammen? — Beweisnot
3. Bottom-up Aufbau der Lösungstabelle — einfach
4. Rekonstruktion der Lösung — einfach
5. Verfeinerungen:
   Platz sparen, Cache-effizient, Parallelisierung — Standard-Trickkiste

Anwendungen dynamischer Programmierung

- Bellman-Ford-Alg. für kürzeste Wege — Teilpfade
- Edit distance/approx. string matching — Algorithmen II?
- Verkettete Matrixmultiplikation — Übung?
- Rucksackproblem — Gegenstände $1..i$ füllen Teil des Rucksacks
- Geld wechseln — Übung?

Gegenbeispiel: Teilproblemeigenschaft

Angenommen, die schnellste Strategie für 20 Runden auf dem Hockenheimring verbraucht den Treibstoff vollständig.

$\Rightarrow$

Keine gute Teilstrategie für 21 Runden.

Frage: Wie kann man "constrained shortest path" trotzdem mittels dynamischer Programmierung modellieren?

Gegenbeispiel: Austauschbarkeit

Optimale Graphfärbungen sind nicht austauschbar.