Technische Universität Braunschweig

Algorithmen und Datenstrukturen II

4. Vorlesung

Linda Kleist, 15.05.2019

Lösen von Subset Sum - Wiederholung

Problem 5 (Subset Sum)

Gegeben: - Objekte $\{1, \ldots, n\}$, je mit Größe z_i
- Zielgröße Z

Gesucht: Eine Menge $S \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in S} z_i = Z$.

Wir definieren eine Hilfsfunktion:

$$\mathcal{S} \colon \{0, \ldots, Z\} \times \{0, \ldots, n\} \to \{0, 1\}$$

$$\mathcal{S}(x, i) := \begin{cases} 1, & \text{falls Teilmenge von } z_1, \ldots, z_i \text{ Summe } x \text{ ergibt} \\ 0, & \text{sonst.} \end{cases}$$

Lösen von Subset Sum - Wiederholung

Algorithmus 3 (Dynamic Program für SUBSET SUM)

Eingabe: Zahlen $z_1, \ldots, z_n, Z$
Ausgabe: 1, falls $S \subseteq \{1, \ldots, n\}$ existiert mit $\sum_{i \in S} z_i = Z$; 0, sonst.

1: $\mathcal{S}(0,0) := 1$
2: **for** $(x = 1)$ to Z **do**
3: $\quad \mathcal{S}(x,0) := 0;$
4: **for** $(i = 1)$ to n **do**
5: $\quad$ **for** $(x = 0)$ to $(z_i - 1)$ **do**
6: $\quad\quad \mathcal{S}(x,i) := \mathcal{S}(x,i-1);$
7: $\quad$ **for** $(x = z_i)$ to Z **do**
8: $\quad\quad$ **if** $(\mathcal{S}(x,i-1) = 1)$ OR $(\mathcal{S}(x-z_i,i-1) = 1)$ **then**
9: $\quad\quad\quad \mathcal{S}(x,i) := 1;$
10: $\quad\quad$ **else**
11: $\quad\quad\quad \mathcal{S}(x,i) := 0;$
12: **return** $\mathcal{S}(Z,n)$

$$\mathcal{S}(x,i) := \max\{\mathcal{S}(x,i-1), \mathcal{S}(x-z_i,i-1)\}$$

Lösen von Subset Sum - Wiederholung

Satz 3

Algorithmus 3 entscheidet ob SUBSET SUM eine Lösung hat, die Laufzeit beträgt $O(nZ)$.

Beweis.

Tafel... □

Verallgemeinerung für Maximum Knapsack

Problem 2 (Maximum Knapsack)

Gegeben: - Objekte $\{O_1, ..., O_n\}$, je mit Größe z_i und Gewinn p_i
- Größenschranke Z

Gesucht: Eine Menge $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und

$\sum_{i \in S} p_i$ maximal.

Beispiel

Gegeben sei $Z = 10$ und fünf Objekte mit den Werten:

i	1	2	3	4	5
z_i	5	7	12	2	3
p_i	3	4	10	5	2

Verallgemeinerung für Maximum Knapsack

Problem 2 (Maximum Knapsack)

Gegeben: - Objekte $\{O_1, ..., O_n\}$, je mit Größe z_i und Gewinn p_i
- Größenschranke Z

Gesucht: Eine Menge $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und

$\sum_{i \in S} p_i$ maximal.

Wir bezeichnen die Menge aller Teilmengen von $[i]$ mit Gesamtgröße $\leq x$ mit $\mathfrak{S}_{x,i} := \{S \subseteq \{1, \ldots, i\} \mid \sum_{k \in S} z_k \leq x\}$.

$$\mathcal{P}\colon \{0, \ldots, Z\} \times \{0, \ldots, n\} \to \mathbb{R} \text{ mit}$$

$$\mathcal{P}(x, i) := \max \left\{ \sum_{k \in S} p_k \mid S \in \mathfrak{S}_{x,i} \right\}$$

Verallgemeinerung für Maximum Knapsack

Problem 2 (Maximum Knapsack)

Gegeben: - Objekte $\{O_1, ..., O_n\}$, je mit Größe z_i und Gewinn p_i
- Größenschranke Z

Gesucht: Eine Menge $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und

$\sum_{i \in S} p_i$ maximal.

Es gilt:

$$\mathcal{P}(x, i) = \begin{cases} 0, & \text{falls } i = 0, \\ \mathcal{P}(x, i-1), & \text{falls } i > 0 \text{ und } z_i > x, \\ \max\{\mathcal{P}(x, i-1), \mathcal{P}(x - z_i, i-1) + p_i\}, & \text{sonst.} \end{cases}$$

Lösen von Maximum Knapsack

Algorithmus 4 (Dynamic Program für MAXIMUM KNAPSACK)

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $\max_{S \in \mathfrak{S}} \sum_{i \in S} p_i$ wobei $\mathfrak{S} := \{S \subseteq \{1, \ldots, n\} \mid \sum_{i \in S} z_i \leq Z\}$.

```
1:  for (x = 0) to Z do
2:      P(x, 0) := 0
3:  for (i = 1) to n do
4:      for (x = 0) to (z_i − 1) do
5:          P(x, i) := P(x, i − 1)
6:      for (x = z_i) to Z do
7:          if ((P(x − z_i, i − 1) + p_i) > P(x, i − 1)) then
8:              P(x, i) := P(x − z_i, i − 1) + p_i
9:          else
10:             P(x, i) := P(x, i − 1)
11: return P(Z, n)
```

Lösen von Maximum Knapsack

Satz 4

Algorithmus 4 berechnet den optimalen Lösungswert für MAXIMUM KNAPSACK in einer Laufzeit $O(nZ)$.

Beweis.

Gute Übung. (Analog zum Beweis von Satz 3.) □

Bemerkungen:

- Die Laufzeit ist pseudopolynomiell (Z kann groß sein und in geeigneter Kodierung $O(\log Z)$ Bits benötigen).
- Die Tabelle benötigt viel Speicherplatz (obwohl zur Berechnung einer Zeile nur die Vorherige benötigt wird).

Lösen von Maximum Knapsack – sparsam

Algorithmus 5 (sparsames Dynamic Program für MAXIMUM KNAPSACK)

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $\max_{S \in \mathcal{S}} \sum_{i \in S} p_i$ wobei $\mathcal{S} := \{S \subseteq \{1, \ldots, n\} \mid \sum_{i \in S} z_i \leq Z\}$.

1: **for** $(x = 0)$ to Z **do**
2: $\quad \mathcal{P}(x) := 0$
3: **for** $(i = 1)$ to n **do**
4: $\quad$ **for** $(x = Z)$ downto z_i **do**
5: $\quad\quad$ **if** $((\mathcal{P}(x - z_i) + p_i) > \mathcal{P}(x))$ **then**
6: $\quad\quad\quad \mathcal{P}(x) := \mathcal{P}(x - z_i) + p_i$
7: **return** $\mathcal{P}(Z)$

Satz 5

Algorithmus 5 berechnet den optimalen Lösungswert für MAXIMUM KNAPSACK in einer Laufzeit von $O(nZ)$.

Berechnen einer Lösung

- 1. Idee: Teillösungen speichern
  - Sei $\mathcal{L}(x, i)$ sei die Indexmenge einer Lösung zu $\mathcal{P}(x, i)$.
  - Es gilt:

$$\mathcal{L}(x,i) := \begin{cases} \emptyset & \text{falls } i = 0, \\ \mathcal{L}(x - z_i, i-1) \cup \{i\} & \text{falls } i > 0, x \geq z_i \text{ und} \\ & (\mathcal{P}(x - z_i, i-1) + p_i) > \mathcal{P}(x, i-1)), \\ \mathcal{L}(x, i-1) & \text{sonst.} \end{cases}$$

- 2. Idee: merken, ob Objekt zur Verbesserung geführt hat
  - für $i > 0$ setzen wir $V(x,i) := \begin{cases} 0, & \text{falls } \mathcal{P}(x,i) = \mathcal{P}(x,i-1), \\ 1, & \text{sonst.} \end{cases}$
  - Konstruktion von $\mathcal{L}(Z, n)$ im Nachhinein

Dynamische Programmierung

- Optimallösung setzt sich aus optimalen Teillösungen zusammen
  $\rightarrow$ optimale Substruktur
- Vereinfachung durch Zurückführen auf kleinere/einfachere Probleme

Entwerfen eines Dynamischen Programms

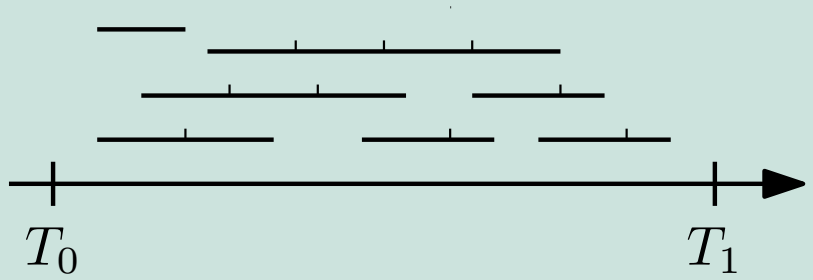
- Optimierungskritierium definieren
- Teilprobleme oder Hilfsprobleme definieren
- Rekursionsgleichung aufstellen
- geeignete Datenstruktur finden
- Abhängigkeiten identifizieren, Auswertungsreihenfolge bestimmen
- Algorithmus schreiben

Dynamische Programmierung

Beispiel: Hörsaal-Auslastung

Gegeben: - Zeitspanne (T_0, T_1)
- Veranstaltungen mit Start- und Endzeiten (s_i, e_i), $i \in \{1, \ldots, n\}$

Gesucht: Auswahl von disjunkten Veranstaltungen, so dass Hörsaal möglichst viel belegt



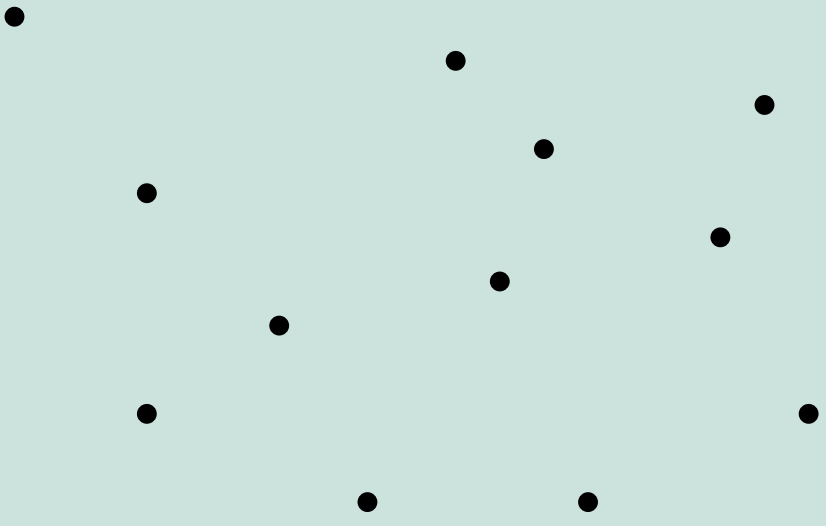
Frage: Was sind hilfreiche Teilprobleme?

Dynamisches Programm für TSP

Beispiel: Euklidisches Rundreise-Problem

Gegeben: Menge von Punkten in der Ebene

Gesucht: kürzeste Rundreise, die alle Punkte besucht.

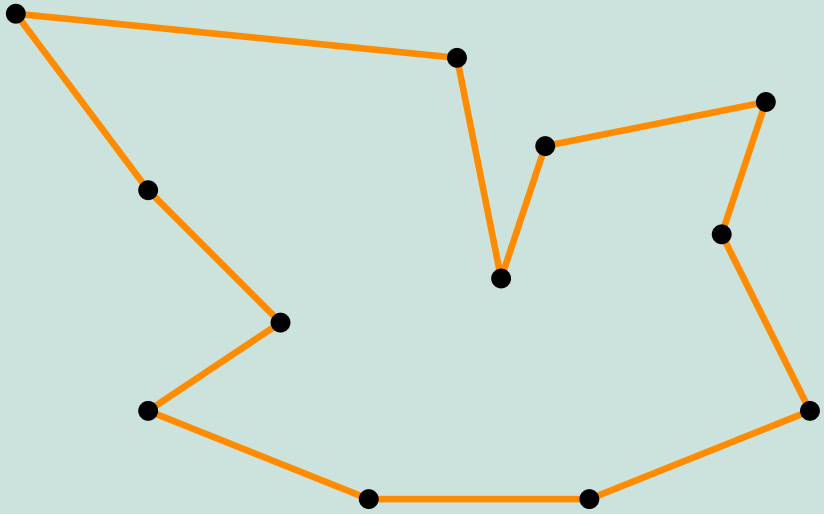


Dynamisches Programm für TSP

Beispiel: Euklidisches Rundreise-Problem

Gegeben: Menge von Punkten in der Ebene

Gesucht: kürzeste Rundreise, die alle Punkte besucht.

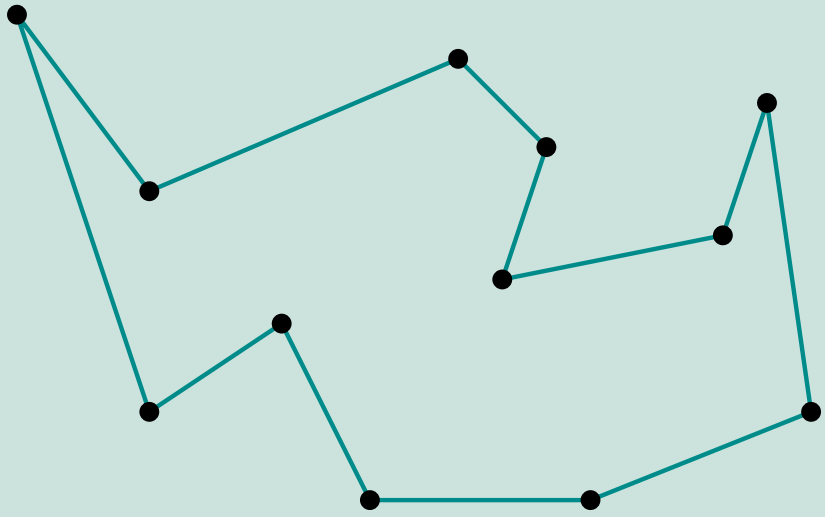


Dynamisches Programm für TSP

Beispiel: Euklidisches Rundreise-Problem

Gegeben: Menge von Punkten in der Ebene

Gesucht: kürzeste Rundreise, die alle Punkte besucht.

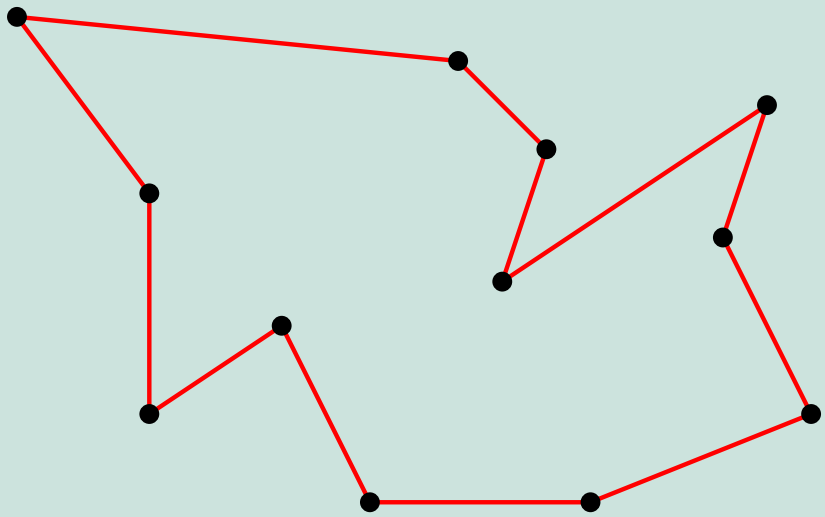


Dynamisches Programm für TSP

Beispiel: Euklidisches Rundreise-Problem

Gegeben: Menge von Punkten in der Ebene

Gesucht: kürzeste Rundreise, die alle Punkte besucht.



Frage: Was sind hilfreiche Teilprobleme?