

Komplexitätstheorie: Zahlenprobleme

Martin Dietzfelbinger

25. Mai 2009

Kapitel 2

Grundlegende NP-vollständige Probleme

2.2 Einfache Zahlenprobleme

Suchproblem RUCKSACK*:

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n ,

(Rucksack mit) Volumen b .

Suchproblem RUCKSACK*:

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n ,

(Rucksack mit) Volumen b .

Aufgabe: Finde eine **Auswahl** der Objekte, die den Rucksack **exakt füllen!**

Suchproblem RUCKSACK*:

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n ,

(Rucksack mit) Volumen b .

Aufgabe: Finde eine **Auswahl** der Objekte, die den Rucksack **exakt füllen!**

Finde $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = b$.

Suchproblem RUCKSACK*:

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n ,

(Rucksack mit) Volumen b .

Aufgabe: Finde eine **Auswahl** der Objekte, die den Rucksack **exakt füllen!**

Finde $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = b$.

Entscheidungsvariante RUCKSACK*: Gibt es eine solche Auswahl?

Formalisierung als Sprache:

$$\text{RUCKSACK}^* = \left\{ (a_1, \dots, a_n, b) \mid n \geq 0, a_1, \dots, a_n, b \in \mathbb{N}, \right. \\ \left. \exists I \subseteq \{1, \dots, n\} : \sum_{i \in I} a_i = b \right\}.$$

Satz 2.2.1 RUCKSACK* ist **NP**-vollständig.

Satz 2.2.1 RUCKSACK* ist **NP**-vollständig.

Beweis:

Satz 2.2.1 RUCKSACK* ist **NP**-vollständig.

Beweis:

(i) RUCKSACK* \in **NP**: Entscheidungsvariante eines **NP**-Suchproblems.

Satz 2.2.1 RUCKSACK* ist **NP**-vollständig.

Beweis:

(i) RUCKSACK* \in **NP**: Entscheidungsvariante eines **NP**-Suchproblems.

(ii) RUCKSACK* ist **NP**-schwer:
Reduktionsmethode.

Wir zeigen: $3\text{-SAT} \leq_p \text{RUCKSACK}^*$.

Satz 2.2.1 RUCKSACK* ist **NP**-vollständig.

Beweis:

(i) RUCKSACK* \in **NP**: Entscheidungsvariante eines **NP**-Suchproblems.

(ii) RUCKSACK* ist **NP**-schwer:

Reduktionsmethode.

Wir zeigen: $3\text{-SAT} \leq_p \text{RUCKSACK}^*$.

Das heißt: Wir finden Reduktionsfunktion

$$f: \{\varphi \mid \varphi \text{ KNF-Formel}\} \rightarrow \{(a_1, \dots, a_n, b) \mid a_1, \dots, a_n, b \in \mathbb{N}\}$$

mit $f \in \mathbf{FP}$ und, für alle 3-KNF-Formeln φ :

$$\varphi \text{ erfüllbar} \Leftrightarrow f(\varphi) = x_\varphi \in \text{RUCKSACK}^*.$$

Wenn $\varphi = C_1 \wedge \dots \wedge C_r$ 3-KNF-Formel ist, die genau k verschiedene Variablen hat,

(Beispiel:

$$(X_{19} \vee \overline{X}_{404} \vee X_7) \wedge (\overline{X}_{15} \vee X_{99} \vee X_{404}) \wedge (X_{19} \vee \overline{X}_{99} \vee X_{60}))$$

Wenn $\varphi = C_1 \wedge \dots \wedge C_r$ 3-KNF-Formel ist, die genau k verschiedene Variablen hat,

(Beispiel:

$$(X_{19} \vee \overline{X}_{404} \vee X_7) \wedge (\overline{X}_{15} \vee X_{99} \vee X_{404}) \wedge (X_{19} \vee \overline{X}_{99} \vee X_{60}))$$

benennen wir die Variablen in $X_1 \dots, X_k$ um (alle Variablen kommen vor):

$$(X_1 \vee \overline{X}_2 \vee X_3) \wedge (\overline{X}_4 \vee X_5 \vee X_2) \wedge (X_1 \vee \overline{X}_5 \vee X_6),$$

Wenn $\varphi = C_1 \wedge \dots \wedge C_r$ 3-KNF-Formel ist, die genau k verschiedene Variablen hat,

(Beispiel:

$$(X_{19} \vee \overline{X}_{404} \vee X_7) \wedge (\overline{X}_{15} \vee X_{99} \vee X_{404}) \wedge (X_{19} \vee \overline{X}_{99} \vee X_{60}))$$

benennen wir die Variablen in $X_1 \dots, X_k$ um (alle Variablen kommen vor):

$$(X_1 \vee \overline{X}_2 \vee X_3) \wedge (\overline{X}_4 \vee X_5 \vee X_2) \wedge (X_1 \vee \overline{X}_5 \vee X_6),$$

ohne dass sich an der Erfüllbarkeit etwas ändert.

$f(\varphi) = x_\varphi = (a_1, \dots, a_n, b)$ hat folgendes Format:

$n = 2k + 2r$ und

die Zahlen a_1, \dots, a_n, b haben $k + r$ Ziffern.

(Dezimal, oder bequemer: oktal).

$f(\varphi) = x_\varphi = (a_1, \dots, a_n, b)$ hat folgendes Format:

$n = 2k + 2r$ und

die Zahlen a_1, \dots, a_n, b haben $k + r$ Ziffern.

(Dezimal, oder bequemer: oktal).

Eine Ziffer für jede Variable und eine für jede Klausel.

Die Zahlen a_1, \dots, a_n sind in 4 Gruppen eingeteilt:

c_1, \dots, c_k (für $v(X_1) = 1, \dots, v(X_k) = 1$)

d_1, \dots, d_k (für $v(X_1) = 0, \dots, v(X_k) = 0$)

Die Zahlen a_1, \dots, a_n sind in 4 Gruppen eingeteilt:

c_1, \dots, c_k (für $v(X_1) = 1, \dots, v(X_k) = 1$)

d_1, \dots, d_k (für $v(X_1) = 0, \dots, v(X_k) = 0$)

p_1, \dots, p_r (zum „Auffüllen“)

q_1, \dots, q_r (zum „Auffüllen“)

Die Zahlen a_1, \dots, a_n sind in 4 Gruppen eingeteilt:

c_1, \dots, c_k (für $v(X_1) = 1, \dots, v(X_k) = 1$)

d_1, \dots, d_k (für $v(X_1) = 0, \dots, v(X_k) = 0$)

p_1, \dots, p_r (zum „Auffüllen“)

q_1, \dots, q_r (zum „Auffüllen“)

Also: $n = 2k + 2r$

Die Zahlen a_1, \dots, a_n sind in 4 Gruppen eingeteilt:

c_1, \dots, c_k (für $v(X_1) = 1, \dots, v(X_k) = 1$)

d_1, \dots, d_k (für $v(X_1) = 0, \dots, v(X_k) = 0$)

p_1, \dots, p_r (zum „Auffüllen“)

q_1, \dots, q_r (zum „Auffüllen“)

Also: $n = 2k + 2r$

und $(a_1, \dots, a_n, b) = (c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r, b)$.

$d_i, 1 \leq i \leq k:$

Position i gleich 1;

Position $1 \leq i' \leq k, i' \neq i$, gleich 0;

Position $k + j$ gleich 1 falls \overline{X}_i in C_j vorkommt,
0 sonst.

$d_i, 1 \leq i \leq k:$

Position i gleich 1;

Position $1 \leq i' \leq k, i' \neq i$, gleich 0;

Position $k + j$ gleich 1 falls \overline{X}_i in C_j vorkommt,
0 sonst.

$p_j, 1 \leq j \leq r:$

Position $k + j$ gleich 2, alle anderen sind 0.

$q_j, 1 \leq j \leq r:$

Position $k + j$ gleich 1, alle anderen sind 0.

$d_i, 1 \leq i \leq k:$

Position i gleich 1;

Position $1 \leq i' \leq k, i' \neq i$, gleich 0;

Position $k + j$ gleich 1 falls \overline{X}_i in C_j vorkommt,
0 sonst.

$p_j, 1 \leq j \leq r:$

Position $k + j$ gleich 2, alle anderen sind 0.

$q_j, 1 \leq j \leq r:$

Position $k + j$ gleich 1, alle anderen sind 0.

Klar: Aus $\varphi = C_1 \wedge \dots \wedge C_r$ ist

$f(\varphi) = x_\varphi = (c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r, b)$

leicht (in Polynomialzeit) zu berechnen.

Beispiel:

Beispiel: $\varphi = (X_1 \vee \overline{X}_2 \vee X_3) \wedge (X_2 \vee \overline{X}_3 \vee \overline{X}_4) \wedge (\overline{X}_2 \vee X_3 \vee X_4)$

Beispiel: $\varphi = (X_1 \vee \bar{X}_2 \vee X_3) \wedge (X_2 \vee \bar{X}_3 \vee \bar{X}_4) \wedge (\bar{X}_2 \vee X_3 \vee X_4)$

Komponenten von $f(\varphi) = x_\varphi$:

$$c_1 = 1000\ 100$$

$$c_2 = 0100\ 010$$

$$c_3 = 0010\ 101$$

$$c_4 = 0001\ 001$$

$$p_1 = 0000\ 200$$

$$p_2 = 0000\ 020$$

$$p_3 = 0000\ 002$$

$$b = 1111\ 444.$$

$$d_1 = 1000\ 000$$

$$d_2 = 0100\ 101$$

$$d_3 = 0010\ 010$$

$$d_4 = 0001\ 010$$

$$q_1 = 0000\ 100$$

$$q_2 = 0000\ 010$$

$$q_3 = 0000\ 001$$

Beispiel: $\varphi = (X_1 \vee \bar{X}_2 \vee X_3) \wedge (X_2 \vee \bar{X}_3 \vee \bar{X}_4) \wedge (\bar{X}_2 \vee X_3 \vee X_4)$

Komponenten von $f(\varphi) = x_\varphi$:

c_1	$=$	1000 100	d_1	$=$	1000 000
c_2	$=$	0100 010	d_2	$=$	0100 101
c_3	$=$	0010 101	d_3	$=$	0010 010
c_4	$=$	0001 001	d_4	$=$	0001 010
p_1	$=$	0000 200	q_1	$=$	0000 100
p_2	$=$	0000 020	q_2	$=$	0000 010
p_3	$=$	0000 002	q_3	$=$	0000 001
b	$=$	1111 444.			

Ebenfalls leicht: Übersetzen (aus Oktal-) in Binärdarstellung.

Zu zeigen:

φ ist erfüllbar $\Leftrightarrow x_\varphi \in \text{RUCKSACK}^*$.

Zu zeigen:

$$\varphi \text{ ist erfüllbar} \Leftrightarrow x_\varphi \in \text{RUCKSACK}^*.$$

„Aus einer erfüllenden Belegung kann man eine Auswahl der Gegenstände mit Summe b erhalten und umgekehrt.“

Zu zeigen:

$$\varphi \text{ ist erfüllbar} \Leftrightarrow x_\varphi \in \text{RUCKSACK}^*.$$

„Aus einer erfüllenden Belegung kann man eine Auswahl der Gegenstände mit Summe b erhalten und umgekehrt.“

“ \Rightarrow “: Sei v eine erfüllende Belegung für φ .

Zu zeigen:

$$\varphi \text{ ist erfüllbar} \Leftrightarrow x_\varphi \in \text{RUCKSACK}^*.$$

„Aus einer erfüllenden Belegung kann man eine Auswahl der Gegenstände mit Summe b erhalten und umgekehrt.“

“ \Rightarrow “: Sei v eine erfüllende Belegung für φ .

Wir wählen zuerst aus $c_1, \dots, c_k, d_1, \dots, d_k$ folgende Zahlen:

Zu zeigen:

$$\varphi \text{ ist erfüllbar} \Leftrightarrow x_\varphi \in \text{RUCKSACK}^*.$$

„Aus einer erfüllenden Belegung kann man eine Auswahl der Gegenstände mit Summe b erhalten und umgekehrt.“

“ \Rightarrow “: Sei v eine erfüllende Belegung für φ .

Wir wählen zuerst aus $c_1, \dots, c_k, d_1, \dots, d_k$ folgende Zahlen:

c_i falls $v(X_i) = 1$, und d_i falls $v(X_i) = 0$.

Zu zeigen:

$$\varphi \text{ ist erfüllbar} \Leftrightarrow x_\varphi \in \text{RUCKSACK}^*.$$

„Aus einer erfüllenden Belegung kann man eine Auswahl der Gegenstände mit Summe b erhalten und umgekehrt.“

“ \Rightarrow “: Sei v eine erfüllende Belegung für φ .

Wir wählen zuerst aus $c_1, \dots, c_k, d_1, \dots, d_k$ folgende Zahlen:

c_i falls $v(X_i) = 1$, und d_i falls $v(X_i) = 0$.

Die Summe der ausgewählten Zahlen im Block der ersten k Ziffern ist $111 \dots 111$.

Beispiel:

$$\varphi = (X_1 \vee \overline{X_2} \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee \overline{X_4}) \wedge (\overline{X_2} \vee X_3 \vee X_4)$$

$v(X_1) = v(X_2) = v(X_4) = 1$ und $v(X_3) = 0$ ist erfüllend.

Beispiel:

$$\varphi = (X_1 \vee \overline{X_2} \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee \overline{X_4}) \wedge (\overline{X_2} \vee X_3 \vee X_4)$$

$v(X_1) = v(X_2) = v(X_4) = 1$ und $v(X_3) = 0$ ist erfüllend.

Wir wählen: c_1, c_2, c_4, d_3 .

Summe: 1111 121

Beispiel:

$$\varphi = (X_1 \vee \overline{X_2} \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee \overline{X_4}) \wedge (\overline{X_2} \vee X_3 \vee X_4)$$

$v(X_1) = v(X_2) = v(X_4) = 1$ und $v(X_3) = 0$ ist erfüllend.

Wir wählen: c_1, c_2, c_4, d_3 .

Summe: 1111 121

Wir wählen weiter: p_1, q_1, p_2, p_3, q_3 .

Beispiel:

$$\varphi = (X_1 \vee \overline{X_2} \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee \overline{X_4}) \wedge (\overline{X_2} \vee X_3 \vee X_4)$$

$v(X_1) = v(X_2) = v(X_4) = 1$ und $v(X_3) = 0$ ist erfüllend.

Wir wählen: c_1, c_2, c_4, d_3 .

Summe: 1111 121

Wir wählen weiter: p_1, q_1, p_2, p_3, q_3 .

Liefert Summe $b = 1111 444$.

In Ziffern im zweiten Block können als Summe der ausgewählten c_i bzw. d_i 1, 2 oder 3 erscheinen:

v ist erfüllende Belegung, also gibt es für jede Klausel C_j eine Variable X_i mit der „richtigen“ Belegung, so dass das gewählte c_i bzw. d_i an Stelle $k + j$ eine 1 hat.

Jede Klausel hat nur 3 Literale, also kann keine Position > 3 sein.

In Ziffern im zweiten Block können als Summe der ausgewählten c_i bzw. d_i 1, 2 oder 3 erscheinen:

v ist erfüllende Belegung, also gibt es für jede Klausel C_j eine Variable X_i mit der „richtigen“ Belegung, so dass das gewählte c_i bzw. d_i an Stelle $k + j$ eine 1 hat.

Jede Klausel hat nur 3 Literale, also kann keine Position > 3 sein.

Mit einer passenden Auswahl aus $p_j, q_j, 1 \leq i \leq r$ können wir jede Position auf 4 auffüllen und so die Summe $b = 111 \dots 11144 \dots 44$ erreichen.

Also **gibt es** eine Auswahl

aus den Zahlen $c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r$,
die sich zu b summiert.

Also **gibt es** eine Auswahl

aus den Zahlen $c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r$,
die sich zu b summiert.

Also ist $x_\varphi \in \text{RUCKSACK}^*$.

Zu zeigen:

φ erfüllbar $\Leftrightarrow x_\varphi \in \text{RUCKSACK}^*$.

“ \Leftarrow “:

Zu zeigen:

φ erfüllbar $\Leftrightarrow x_\varphi \in \text{RUCKSACK}^*$.

“ \Leftarrow “: Sei eine Auswahl aus den Zahlen $c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r$ gegeben, die sich zu b summiert.

Zu zeigen:

φ erfüllbar $\Leftrightarrow x_\varphi \in \text{RUCKSACK}^*$.

“ \Leftarrow “: Sei eine Auswahl aus den Zahlen

$c_1, \dots, c_k, d_1, \dots, d_k, p_1, \dots, p_r, q_1, \dots, q_r$
gegeben, die sich zu b summiert.

Es kann keine Überträge zwischen Ziffernpositionen geben.

(In jeder Ziffernposition ist die Summe maximal 6.)

Daher: Addition funktioniert wie bei Vektoren
(komponentenweise).

Wegen der 1en im ersten Teil von b :

Für jedes i ist c_i oder d_i gewählt, aber nicht beide.

Wegen der 1en im ersten Teil von b :

Für jedes i ist c_i oder d_i gewählt, aber nicht beide.

– Definiere:

$v(X_i) := 1$ falls c_i gewählt und $v(X_i) := 0$ falls d_i gewählt.

Wegen der 1en im ersten Teil von b :

Für jedes i ist c_i oder d_i gewählt, aber nicht beide.

– Definiere:

$v(X_i) := 1$ falls c_i gewählt und $v(X_i) := 0$ falls d_i gewählt.

Diese Belegung v muss jede Klausel C_j von φ erfüllen,

da sonst die Summe 4 in Position $k+j$ nicht erreicht werden kann. \square

Rucksack-Problem: Ein **NP**-Optimierungsproblem.

Rucksack-Problem: Ein **NP**-Optimierungsproblem.

Gegeben: $a_1, \dots, a_m, c_1, \dots, c_m, b \in \mathbb{N}$.

a_1, \dots, a_m : „Volumina“ (von n Gegenständen)

b : Volumenschranke

c_1, \dots, c_m :

„Nutzenwerte“, „Profite“ der n Gegenstände

Rucksack-Problem: Ein **NP**-Optimierungsproblem.

Gegeben: $a_1, \dots, a_m, c_1, \dots, c_m, b \in \mathbb{N}$.

a_1, \dots, a_m : „Volumina“ (von n Gegenständen)

b : Volumenschranke

c_1, \dots, c_m :

„Nutzenwerte“, „Profite“ der n Gegenstände

Gesucht: Eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände

mit $\sum_{i \in I} a_i \leq b$ (Volumenschranke einhalten)

und $\sum_{i \in I} c_i$ möglichst groß.

Entscheidungsproblem (als Sprache): RUCKSACK.

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Beweis:

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Beweis: (i) RUCKSACK \in **NP**: Wie üblich.

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Beweis: (i) RUCKSACK \in **NP**: Wie üblich.

(ii) RUCKSACK ist **NP**-schwer:

Wir zeigen: $\text{RUCKSACK}^* \leq_p \text{RUCKSACK}$.

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Beweis: (i) RUCKSACK \in **NP**: Wie üblich.

(ii) RUCKSACK ist **NP**-schwer:

Wir zeigen: $\text{RUCKSACK}^* \leq_p \text{RUCKSACK}$.

Reduktionsfunktion: $(a_1, \dots, a_n, b) \mapsto (a_1, \dots, a_n, a_1, \dots, a_n, b, b)$.

Entscheidungsproblem (als Sprache): RUCKSACK.

Input: $a_1, \dots, a_m, c_1, \dots, c_m, b, k \in \mathbb{N}$.

Frage: Gibt es eine Auswahl $I \subseteq \{1, \dots, m\}$ der Gegenstände, die $\sum_{i \in I} a_i \leq b$ und $\sum_{i \in I} c_i \geq k$ erfüllen?

Satz 2.2.2 RUCKSACK ist **NP**-vollständig.

Beweis: (i) RUCKSACK \in **NP**: Wie üblich.

(ii) RUCKSACK ist **NP**-schwer:

Wir zeigen: $\text{RUCKSACK}^* \leq_p \text{RUCKSACK}$.

Reduktionsfunktion: $(a_1, \dots, a_n, b) \mapsto (a_1, \dots, a_n, a_1, \dots, a_n, b, b)$.

Verifiziere die Reduktionseigenschaft!

Partition-Problem: Ein **NP**-Suchproblem.

Partition-Problem: Ein **NP**-Suchproblem.

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n .

Partition-Problem: Ein **NP**-Suchproblem.

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n .

Aufgabe: Teile die Objekte in zwei Teilmengen auf, die genau das gleiche Volumen haben.

Partition-Problem: Ein **NP**-Suchproblem.

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n .

Aufgabe: Teile die Objekte in zwei Teilmengen auf, die genau das gleiche Volumen haben.

Formal: Finde $I \subseteq \{1, \dots, n\}$ mit

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i, \text{ falls m\u00f6glich.}$$

(Falls nicht m\u00f6glich, gib „unm\u00f6glich“ aus.)

Partition-Problem: Ein **NP**-Suchproblem.

Gegeben:

(n Objekte mit) Volumina a_1, \dots, a_n .

Aufgabe: Teile die Objekte in zwei Teilmengen auf, die genau das gleiche Volumen haben.

Formal: Finde $I \subseteq \{1, \dots, n\}$ mit

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i, \text{ falls m\u00f6glich.}$$

(Falls nicht m\u00f6glich, gib „unm\u00f6glich“ aus.)

Beispiel: Bei Eingabe $(1, 2, 2, 8, 3, 4)$ w\u00e4re $I = \{3, 4\}$ eine legale Ausgabe. Bei Eingabe $(1, 2, 4, 8, 16)$ w\u00e4re die Ausgabe „unm\u00f6glich“.

Entscheidungsvariante als Sprache:

Entscheidungsvariante als Sprache:

$$\text{PARTITION} = \left\{ (a_1, \dots, a_n, b) \mid a_1, \dots, a_n, b \in \mathbb{N}, \right. \\ \left. \exists I \subseteq \{1, \dots, n\} : \sum_{i \in I} a_i = b \right\}.$$

Entscheidungsvariante als Sprache:

$$\text{PARTITION} = \left\{ (a_1, \dots, a_n, b) \mid a_1, \dots, a_n, b \in \mathbb{N}, \right. \\ \left. \exists I \subseteq \{1, \dots, n\} : \sum_{i \in I} a_i = b \right\}.$$

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Beweis:

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Beweis:

(i) PARTITION \in **NP**: Wie üblich.

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Beweis:

(i) PARTITION \in **NP**: Wie üblich.

(ii) PARTITION ist **NP**-schwer:

Wir zeigen: RUCKSACK* \leq_p PARTITION..

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Beweis:

(i) PARTITION \in **NP**: Wie üblich.

(ii) PARTITION ist **NP**-schwer:

Wir zeigen: RUCKSACK* \leq_p PARTITION..

Reduktionsfunktion:

$$f((a_1, \dots, a_n, b)) = (a_1, \dots, a_n, s + 1, 2b + 1),$$

für $s = \sum_{1 \leq i \leq n} a_i$.

Satz 2.2.3 PARTITION ist **NP**-vollständig.

Beweis:

(i) PARTITION \in **NP**: Wie üblich.

(ii) PARTITION ist **NP**-schwer:

Wir zeigen: RUCKSACK* \leq_p PARTITION..

Reduktionsfunktion:

$$f((a_1, \dots, a_n, b)) = (a_1, \dots, a_n, s + 1, 2b + 1),$$

für $s = \sum_{1 \leq i \leq n} a_i$.

Zu zeigen:

$$(a_1, \dots, a_n, b) \in \text{RUCKSACK}^*$$

$$\Leftrightarrow (a_1, \dots, a_n, s + 1, 2b + 1) \in \text{PARTITION}.$$

Binpacking-Problem: MINBINPACKING

Ein **NP**-Optimierungsproblem.

Binpacking-Problem: MINBINPACKING

Ein **NP**-Optimierungsproblem.

Gegeben: (n Objekte mit)

Volumina a_1, \dots, a_n , Behälterkapazität b .

Aufgabe: Teile die Objekte auf **möglichst wenige** Behälter auf, so dass in keinem Behälter Objekte mit Gesamtvolumen größer als b sind.

MINBINPACKING Formal:

Gegeben: $a_1, \dots, a_n, b \in \mathbb{N}$.

Aufgabe: Finde $\varrho: \{1, \dots, m\} \rightarrow \{1, \dots, k\}$ mit

$$\forall j \in \{1, \dots, k\}: \sum_{\substack{1 \leq i \leq n \\ \varrho(i)=j}} a_i \leq b,$$

MINBINPACKING Formal:

Gegeben: $a_1, \dots, a_n, b \in \mathbb{N}$.

Aufgabe: Finde $\varrho: \{1, \dots, m\} \rightarrow \{1, \dots, k\}$ mit

$$\forall j \in \{1, \dots, k\}: \sum_{\substack{1 \leq i \leq n \\ \varrho(i)=j}} a_i \leq b,$$

so dass k möglichst klein ist.

Entscheidungsvariante als Sprache:

$$\text{BINPACKING} = \left\{ (a_1, \dots, a_n, b, k) \mid a_1, \dots, a_n, b, k \in \mathbb{N}, \right. \\ \left. \exists \varrho: \{1, \dots, n\} \rightarrow \{1, \dots, k\} : \right. \\ \left. \forall j \in \{1, \dots, k\} : \sum_{\substack{1 \leq i \leq n \\ \varrho(i)=j}} a_i \leq b \right\}.$$

Entscheidungsvariante als Sprache:

$$\text{BINPACKING} = \left\{ (a_1, \dots, a_n, b, k) \mid a_1, \dots, a_n, b, k \in \mathbb{N}, \right. \\ \left. \begin{array}{l} \exists \varrho: \{1, \dots, n\} \rightarrow \{1, \dots, k\}: \\ \forall j \in \{1, \dots, k\} : \sum_{\substack{1 \leq i \leq n \\ \varrho(i)=j}} a_i \leq b \end{array} \right\}.$$

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Beweis:

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Beweis:

(i) BINPACKING \in **NP**: Wie üblich.

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Beweis:

(i) BINPACKING \in **NP**: Wie üblich.

(ii) BINPACKING ist **NP**-schwer:

PARTITION \leq_p BINPACKING.

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Beweis:

(i) BINPACKING \in **NP**: Wie üblich.

(ii) BINPACKING ist **NP**-schwer:

PARTITION \leq_p BINPACKING.

Reduktionsfunktion:

2 Behälter, Kapazität $s = \sum_{1 \leq i \leq n} a_i$.

Satz 2.2.4 BINPACKING ist **NP**-vollständig.

Beweis:

(i) BINPACKING \in **NP**: Wie üblich.

(ii) BINPACKING ist **NP**-schwer:

PARTITION \leq_p BINPACKING.

Reduktionsfunktion:

2 Behälter, Kapazität $s = \sum_{1 \leq i \leq n} a_i$.

Gegenstände verdoppeln!

$$f((a_1, \dots, a_n)) = (2a_1, \dots, 2a_n, s, 2).$$

Zu zeigen:

$$(a_1, \dots, a_n) \in \text{PARTITION}$$
$$\Leftrightarrow$$
$$(2a_1, \dots, 2a_n, s, 2) \in \text{BINPACKING}.$$