Spezialfall davon:

PROBLEM 1.9 ("PARTITION")

Gegeben: n Objekte $1, \dots, n$ mit jeweils Größe z_i

Gesucht: $S \subseteq \{1, \dots, n\}$ mit

$$\sum_{i \in S} z_i = \sum_{i \notin S} z_i$$

1.2 Ein Beispiel

Beispiel 1.10

7, 13, 17, 20, 29, 31, 31, 35, 57

(Man erinnere sich an die allererste AuDI-Vorlesung!)

Gesamtsumme: 240

Also: Gibt es $S \subseteq \{1, \dots, 9\}$ mit $\sum_{i \in S} z_i = 120$?

Antwort: NEIN!

(Damals:

(7), 13, 17, 24, (29), (31), 31, 35, (53))

Wie beweist man die Nichtexistenz?

Ziel: 120

Betrachte: 57 ✓ (o.B.d.A.)

Ziel: 63

Betrachte: 35 ?

Ziel: 28

Betrachte: 31 ×
31 ×
29 ×
25 ? → 8 ×
17 ? → 11 ×
13 ? → 15 ×
7 ? → 21 ×

Geht nicht!

Betrachte: 31 ?

Ziel: 32

Betrachte: 31 ×
29 ×
25 → 12 ×
17 → 15 ×
13 ×
7 ×

Geht nicht!

Betrachte: 29 ?

Ziel: 34

Betrachte: 20 → 14 ×
17 → 17 ×
13 ×
7 ×

Geht nicht!

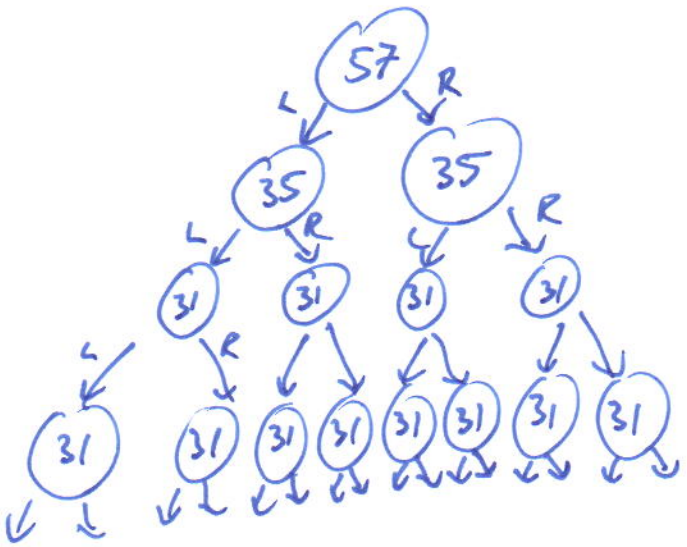
> 120

(Alternative: 35 + 31 + 31 = 97, Ziel 23
geht nicht ...)

Also Idee:
Ausprobieren, systematisch, von größeren zu kleineren Zahlen
↳ Rückwärtsrekursion über verwendete Zahlen und erreichbare Werte!

Vorteil: Funktioniert
Nachteil: Probiert exponentiell viele Teilmengen durch

(Baumdarstellung:



...

↓
2^n Knoten!)

Verbesserungen:

- Schneide Teilbäume ab wenn möglich
  ↳ z.T. Einsparungen, bleibt aber i.d.R. exponentiell

  Später: "Branch and Bound"

- Betrachte nicht Rückwärtsrekursion
  (d.h. Probiere stufenweise aus, reduziere dabei größere Teilinstanzen auf kleinere Teilinstanzen)
  sondern Vorwärtsrekursion:
  Baue größere Lösungen aus kleineren Lösungen auf!

BEISPIEL 1.10A (xkcd 287)

Appetizers	
Mixed Fruit	2.15
French Fries	2.75
Side Salad	3.35
Hot Wings	3.55
Mozzarella Sticks	4.20
Sampler Plate	5.80

"We'd like exactly $15.05 of appetizers, please."

"... exactly? Uh..."

"Here, these papers on the Knapsack problem might help you out."

"Listen, I have six other tables to get to -"

"- as fast as possible, of course. Want something on Traveling Salesman?"

Fragen:
- Wie lautet eine mögliche Kombination?
- Wie gut oder schlecht ist der Greedy-Algorithmus 1.4 für 0-1-Knapsack?
- Wie kann man den Algorithmus erweitern/modifizieren, dass er eine bessere Leistung garantiert?

Große Fragen:
- Wie bekommt man diese Probleme besser in den Griff?
- Wie schwierig kann das werden - bzw. wie lange benötigt das Lösen?

1.3 Dynamisches Programmieren

Prinzip:

1.3.1 Subset Sum

7, 13, 17, 20, 29, 31, 31, 35, 57

→ aufsteigend!

Teste für Zahl x, ob sie mit den ersten i Summanden erzeugt werden kann!

$$S(x,i) := \begin{cases} 0 & \text{falls } x \text{ nicht mit } z_1,\dots,z_i \text{ erzeugt werden kann} \\ 1 & \text{falls } x \text{ erzeugt werden kann} \end{cases}$$

$S(x,0) = 0$ für alle $x > 0$ (aber 1 für $x = 0$)

$$S(x,1) = \begin{cases} 1 & \text{für } x = 7 \\ 0 & \text{sonst} \end{cases}$$

$$S(x,2) = \begin{cases} 1 & \text{für } x = 0, 7; 13, 20 \\ 0 & \text{sonst} \end{cases}$$

$$S(x,3) = \begin{cases} 1 & \text{für } x = 0, 7, 13, 20; 17, 24, 30, 37 \\ 0 & \text{sonst} \end{cases}$$

usw.!

Tabelle:

i \ x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...	240
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1	1	0	0	0	0	0	1	0	...														0
2	1						1							1							1		0
3	...																						

Update-Regel:

(1) $\varphi(x,0) = 0$ für alle $x \in \{1,\dots,Z\}$
$\varphi(0,0) = 1$
} Initialisierung

(2) $\varphi(x, i-1) = 1 \Rightarrow \varphi(x,i) = 1$ $\Leftarrow$ Geht vorher, geht immer noch (ohne z_i)

$\varphi(x - z_i, i-1) = 1 \Rightarrow \varphi(x,i) = 1$ $\Leftarrow$ Geht mit z_i weil $x - z_i$ mit $\{1,\dots,i-1\}$ geht

ALGORITHMUS 1.11 (Dynamic Programming für SUBSET SUM)

Input: $z_1, \dots, z_n$ mit $\sum_{i=1}^{n} z_i = z$

Output: Boolsche Funktion

$$\varphi : \{1, \dots, z\} \times \{1, \dots, n\} \to \{0, 1\}$$
$$(x, i) \mapsto \varphi((x, i))$$

mit $\varphi((x,i)) = \begin{cases} 1 & \text{wenn } x \text{ aus } z_1, \dots, z_i \text{ erzeugbar} \\ 0 & \text{sonst} \end{cases}$

(1) $\varphi(0,0) = 1$;

FOR $(x=1)$ TO z

$\varphi(x,0) = 0$;

(2) FOR $(i=1)$ TO n {

FOR $(x=0)$ TO z {

IF $\left(\left(\varphi(x, i-1) = 1 \right) \text{ OR } \left(\varphi((x - z_i), i-1) = 1 \right) \right)$

$\varphi(x,i) = 1$

ELSE

$\varphi(x,i) = 0$

}

}

(3) RETURN

SATZ 1.12

ALGORITHMUS 1.11 löst das Problem SUBSET SUM.
Die Laufzeit ist $O(Z \cdot n)$.

Beweis:

Vollständige Induktion über n!

$n=0$ ist korrekt initialisiert.

Für den Induktionsschritt nehmen wir an, dass $\varphi(x, i-1)$ für alle x korrekt ist.

Dann betrachten wir im Schritt (2) die Möglichkeiten für die Erzeugung von x mit $z_1, \dots, z_i$:

(I) z_i wird nicht verwendet
$\Rightarrow$ x kann mit $z_1, \dots, z_{i-1}$ erzeugt werden.

(II) z_i wird verwendet
$\Rightarrow$ $x - z_i$ kann mit $z_1, \dots, z_{i-1}$ erzeugt werden.

Genau das macht der Algorithmus in (2).

Daher gilt auch, dass $\varphi(x, i)$ korrekt ist.

□