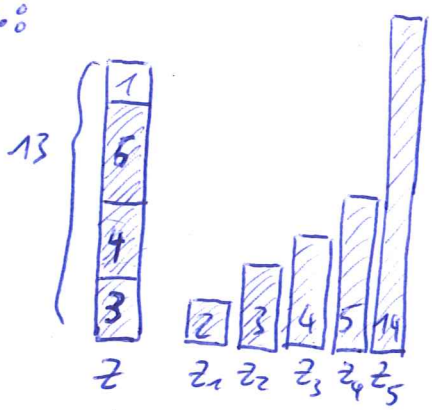
- AuD II: 2. gr. Übung (6.5.2015)
- Gruppeneinteilung (siehe Web)
  → Räume können sich noch ändern

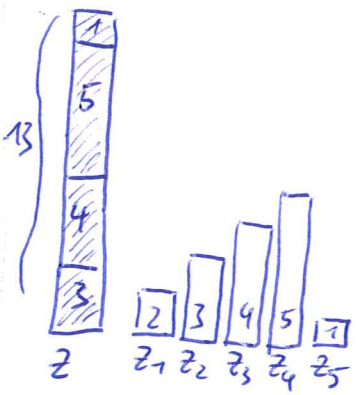
- Problem 1.7 (mehr. Rucksack) mit gleichen Gewichten $z_i = p_i$ vs. Problem 1.8 (Subset Sum)

Gegeben: n Objekte $(1, \dots, n)$ mit jeweils Gewicht z_i für $i \in \{1, \dots, n\}$ und Kapazitätsschranke Z	und Zielgröße Z
Gesucht: $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} z_i \leq Z$ und $\sum_{i \in S} z_i$ maximal	$\sum_{i \in S} z_i = Z$

Bsp.:



opt.: $S = \{2,3,4\}$
$\Rightarrow Z - \sum_{i \in S} z_i$ minimal über alle $S \subseteq \{1, \dots, 5\}$



lösbare Lsg.: $S = \{2,3,4,5\}$
$\Rightarrow Z - \sum_{i \in S} z_i = 0$

Allg.: 1. Für geg. Eingabe $\{z_1, \dots, z_n\}, Z$:

$\sum_{i \in S} z_i \leq Z \iff 0 \leq Z - \sum_{i \in S} z_i$ (notw. Bed. für lösbare Lösung für 1.7, 1.8)

also: $Z - \sum_{i \in S} z_i$ minimal $\iff Z - \sum_{i \in S} z_i = 0$

d.h.: S lösbare Lösung für 1.8 $\iff S$ opt. Lösung für 1.7

Allg.: 2. Für geg. Eingabe $\{z_1, \dots, z_n\}, z$

Es sei $S \subseteq \{1, \dots, n\}$ ort. Lsg für 1.7, dann gilt

1.8 lösbar $\iff z - \sum_{i \in S} z_i = 0$

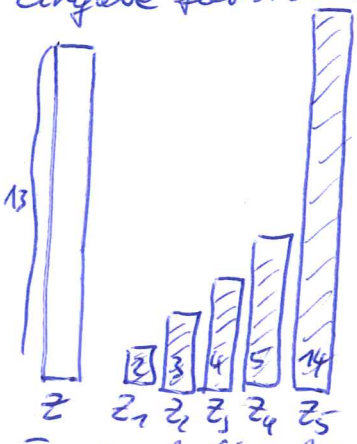
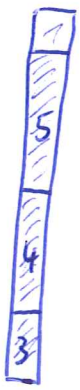
Allg.: 3. Für geg. Eingabe $\{z_1, \dots, z_n\}, z$

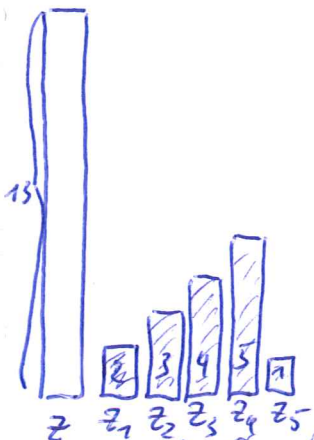
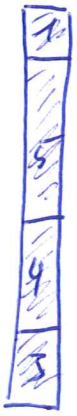
Falls 1.8 nicht lösbar

- Frage: wissen wir etwas über 1.7
- Antwort: <u>Nein</u>, da $z - \sum_{i \in S} z_i$ bel. groß sein kann

Also: 1.7 schwerer als 1.8

Algorithmus, der 1.7 löst, löst auch implizit 1.8

Bsp.: 1.	2.	3.
Eingabe für 1.8:  Füre als Eingabe für 1.7 auf	Berechne ort. Lösung S für 1.7:  $S = \{2,3,4\}$ $\sum_{i \in S} z_i = 3+4+5 = 12 < 13 = z$ $\Rightarrow$ 1.8 nicht lösbar	Ausgabe: 1.8 nicht lösbar

Bsp.: 1.	2.	3.
Eingabe für 1.8:  Füre als Eingabe für 1.7 auf	Berechne ort. Lösung S für 1.7:  $S = \{2,3,4,5\}$ $\sum_{i \in S} z_i = 3+4+5+1 = 13 = 13 = z$ $\Rightarrow$ 1.8 lösbar	Ausgabe: 1.8 lösbar

Wichtig: Eingabe für 1.8 wird in Eingabe für 1.7 transformiert
hier: Transformation trivial

• Problem 1.8 (Subset Sum ~~mit Rucksack gleiche Gewinn-Dichten~~) v.s. Problem 1.9 (Partition)

---

gegeben: n Objekte $(1,\dots,n)$ mit jeweils Größe z_i

und Zielgröße Z

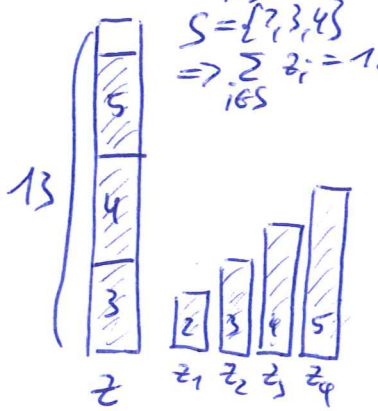
---

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

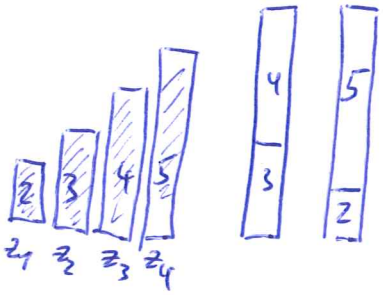
$$\sum_{i\in S} z_i = Z \qquad\qquad \sum_{i\in S} z_i = \sum_{i\notin S} z_i$$

Bsp.:

$S=\{2,3,4\} \Rightarrow \sum_{i\in S} z_i = 12$



$S=\{2,3\} \Rightarrow \sum_{i\in S} z_i = 3+4 = 7 = 2+5 = \sum_{i\notin S} z_i$



• Analog zu „1.8 leichter als 1.7" mit Transformation von Eingabe für 1.8 in Eingabe für 1.7:

Transformation 1.9 → 1.8:

1.9	1.8
Eingabe: $z_1,\dots,z_n$ →	$z_1,\dots,z_n, Z := \frac{z_1+\dots+z_n}{2}$

somit

$\exists S \subseteq \{1,\dots,n\}: \sum_{i\in S} z_i = \sum_{i\notin S} z_i \iff \exists S \subseteq \{1,\dots,n\}: \sum_{i\in S} z_i = Z$

da $z = \frac{\sum_{i=1}^{n} z_i}{2}$

$\Rightarrow \sum_{i \in S} z_i = \frac{\sum_{i=1}^{n} z_i}{2} \Rightarrow \sum_{i \notin S} z_i = \frac{\sum_{i=1}^{n} z_i}{2} = z$

also: Algorithmus für 1.8 löst implizit auch 1.9 (setze $z := \frac{z_1 + \ldots + z_n}{2}$)
1.9 „leichter als" 1.8

Auch: 1.8 „leichter als" 1.9
benötigt Transformation 1.8 auf 1.9:

- Sei $(z_1, \ldots, z_n)$, z Eingabe für 1.8
  $\rightarrow (z_1, \ldots, z_n, z_{n+1}, z_{n+2})$ mit $z_{n+1} := A + z$ und $z_{n+2} := 2A - z$
  für $A := \sum_{i=1}^{n} z_i$

Behauptung: $[(z_1, \ldots, z_n), z$ für 1.8 lösbar$]$
$\Longleftrightarrow$
$[(z_1, \ldots, z_{n+2})$ für 1.9 lösbar$]$

Beweis: „$\Rightarrow$" $\exists S \subseteq \{1, \ldots, n\} : \sum_{i \in S} z_i = z$
$\rightarrow S' := S \cup \{n+2\} \Rightarrow \sum_{i \in S'} = \sum_{i \in S} z_i + 2A - z$
$= z + 2A - z = 2A$

und $\sum_{i \notin S'} z_i = \underbrace{(A - z)}_{\sum_{i=1}^{n} z_i - \sum_{i \in S} z_i} + \underbrace{(A + z)}_{z_{n+1}} = 2A$ ✓

„$\Leftarrow$" $\exists S' \subseteq \{1, \ldots, n+2\} : \sum_{i \in S'} z_i = \sum_{i \notin S'} z_i$

- <u>Entweder</u> z_{n+1} oder z_{n+2} gehört zu S', da
  1. $A + z + 2A - z = 3A$
  2. $\sum_{i=1}^{n+2} z_i = 4A$ d.h. $\sum_{i \in S'} z_i = 2A$ <u>$(< 3A)$</u>
- o.B.d.A: $n+2 \in S'$
  $\rightarrow \sum_{i \in S'} z_i = 2A$
  $\Rightarrow \sum_{i \in S' \setminus \{n+2\}} z_i = 2A - (2A - z) = z$ ✓

Also: 1.8 und 1.9 „gleich schwer"

- Algorithmus für 1.2'
  → Algorithmus für 1.7, 1.8, 1.9
  → Algorithmus 1.15, verwendet dyn. Programmierung
- Dyn. Programmierung
  - Lösen größerer Probleme: Zusammensetzen von Lösungen kleinerer Probleme
  - keine Rekursion
  - Berechne erst alle kleinen Lösungen. Berechne darauf basierend alle nächst größeren Lösungen
    → „Bottom-up"
  - Rekursion hingegen: Betrachte großes Problem zerlege in Teilprobleme und löse diese
    → „Top-down"
  - Größe von Problemen werden über „zahlbare" Dimensionen gemessen (oft > 1)
- Dyn. Programm für max. Rucksack:
  - Funktion $P: \underbrace{\{0,\dots,Z\}}_{\text{1. Dim: restl. Gewichtskapazität}} \times \underbrace{\{0,\dots,n\}}_{\text{2. Dim: } i \hat{=} \text{ Gegenstände } 1,\dots,i \text{ dürfen benutzt werden}} \to \mathbb{N}_{\geq 0}\}$ best möglicher Nutzen, der für jeweiliges Paar von Eingabeparameter erreichbar ist

mit

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

verbleibende Restkap. für Gewichte

Objekte $1,\dots,i$ dürfen verwendet werden

Gegenstand i wird nicht verwendet

Gegenstand i wird verwendet
→ Restkapazität verringert sich um z_i

wenn Gegenstand i verwendet wird, steigt Gesamtnutzen um p_i

heißt z_i nicht mehr (bzgl. x) verwendbar

also: Berechne $P(Z,n)$

- Formulierung durch rekursive Funktion (Top-down), aber Berechnung Bottom-up
  → kleinere Problem (bzgl. x und i) werden zuerst berechnet

- Der dat. Algorithmus iteriert aufsteigend über i und x → zwei geschachtelte for-Schleifen
- Algorithmus iteriert aufsteigend über i und x → zwei geschachtelte for-Schleifen
- Visualisierung d. Vorgehens d. resultierenden Algorithmus
  → Bsp.:

i	1	2	3	4	5	6	7	8	9	10	11
p_i	3	1	2	5	4	3	2	6	2	5	3
z_i	2	4	5	6	2	8	4	1	4	5	9

$Z = 9$

$i \backslash x$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	3	3	3	4	4	4	4
3	0	0	3	3	3	3	3	5	5	5
4	0	0	3	3	3	3	5	5	8	8
5	0	0	4	4	7	7	7	7	9	9
6	0	0	4	4	7	7	7	7	9	9
7	0	0	4	4	7	7	7	7	9	9
8	0	6	6	10	10	13	13	13	13	15
9	0	6	6	10	10	13	13	13	13	15
10	0	6	6	10	10	13	13	13	15	15
11	0	6	6	10	10	13	13	13	15	$\boxed{15}$

initial

→ return 15

- äußere for-Schleife über $i \in \{0, \dots, n\}$
- innere " " " $x \in \{0, \dots, Z\}$

$\Rightarrow$ Laufzeit $\in \Theta(n \cdot Z)$

- Algorithmus für mehr Rucksack indem jeder Algorithmus für SubsetSum und Partition