## Satz 1.16

Algorithmus 1.15 berechnet einen besten Lösungswert für das Rucksackproblem, in $O(nZ)$.

↖ Pseudopolynomiell: Input hat Größe $n \cdot \log Z$

Beweis: Selbst!

Ein Nachteil des Ansatzes:
Am Ende hat man eine ganze Tabelle berechnet, d.h. viel Speicherplatz verbraucht! U.U. interessiert einen aber nur der Optimalwert - und man benötigt jeweils nur die unmittelbar vorangehende Zeile.

Also: Speichere jeweils nur eine Zeile! (Überschreibe bei der Berechnung die vorangehende - bzw. behalte sie)

Hier kann man nicht nur Speicherplatz sparen, sondern auch Codezeilen: (2.1) und (2.22) erübrigen sich.

Damit erhält man:

ALGORITHMUS 1.17 (DP für Knapsack - sparsame)

Input: $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Output: Optimalwert $p^*$ des Knapsackproblems

$$\max \sum_{j=1}^{n} p_j y_j$$

$$\text{mit} \sum_{j=1}^{n} z_j y_j \leq Z$$

$$\text{und} \quad y_j \in \{0,1\}$$

(1) FOR (x=0) TO Z DO {
    P(x) := 0 // Initialisierung
}

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

(2.1) FOR (x=Z) DOWN TO $z_i$ DO {

(2.1.1) IF ( P(x-$z_i$) + $p_i$ > P(x)) THEN
    P(x) := P(x-$z_i$) + $p_i$

}

}

(3) RETURN $p^*$ := P(Z)

Satz 1.18

Algorithmus 1.17 berechnet einen besten Lösungswert für das Rucksackproblem, in $O(n \cdot Z)$.

Beweis: Selbst!

---

Hier nicht dabei:
Wie berechnet man nicht nur den Wert,
sondern auch eine Lösung?

Verschiedene Möglichkeiten:

- Teilmengen „mitschleppen"
- Pointer auf Teillösungen von Zeile zu Zeile
- Andere Möglichkeiten

Wichtig: Laufzeit ⇔ Speicherplatz

Mehr dazu in der Übung!

---

Andere Variante:
DP nicht mit Fokus auf Kosten $z_i$,
sondern auf Werte $p_i$!

Ähnlich, aber nicht Maximieren des Wertes für mögliche Kosten,
sondern Minimieren der Kosten für möglichen Wert!

## 1.4 Branch and Bound

Im Subset-Sum-Beispiel 1.10 haben wir gesehen, dass sich auch beim Enumerieren Arbeit einsparen lässt!

Zur Erinnerung – wir hatten notiert:

- Vorwärtsrekursion statt Rückwärtsrekursion
  ↳ Dynamic Programming

- Schneide Teilbäume ab wenn möglich
  ↳ Branch and Bound
  (verzweige, beschränke)

Das betrachten wir in diesem Teilkapitel!

Grundidee:

(1) Enumeriere die möglichen Teilmengen in einen Enumerationsbaum
(2) Behalte dabei den Zielfunktionswert im Auge:
  - Untere Schranke: Erreichter Zielfunktionswert im ganzen Enumerationsbaum
  - Obere Schranke: Erreichbarer Zielfunktionswert im aktuellen Teilbaum
(3) Beide Schranken sollten einfach und schnell zu berechnen sein
(4) Wenn der erreichbare Wert kleiner bleiben muss als der bereits erreichte, können wir den aktuellen Teilbaum abschneiden

Wir betrachten noch einmal

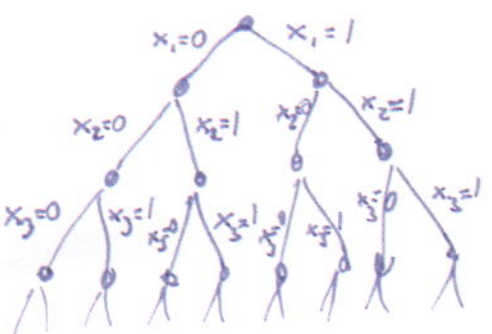
Beispiel 1.14 (Knapsack)

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $z_i$ | 2 | 3 | 6 | 7 | 5 | 9 | 4 |
| $p_i$ | 6 | 5 | 8 | 9 | 6 | 7 | 3 |

$Z = 9$ , $n = 7$

(1) Wie können wir das enumerieren?

- Probiere nacheinander für $i = 1, \dots, 7$ , ob $x_i = 0$ oder $1$
- Gehe die Möglichkeiten baumartig durch:




- Laufe DFS-mäßig durch den Baum; d.h. arbeite die Entscheidungen im Stack ab:
  - Probiere $x_1 = 0$
  - Probiere $x_2 = 0$
  - Probiere $x_3 = 0$
  - :

  | Wenn ein $x_i = 0$ nicht mehr funktioniert, probiere $x_i = 1$ |
  |---|

(2) Welche Untere und obere Schranke haben wir zur Verfügung?

Untere Schranke: Greedy!

- Verwende unter den noch nicht zugewiesenen Objekten in aufsteigendem Kosten/Nutzen-Verhältnis Objekte, solange sie passen.

Hier ohne Fixierte Objekte:

$i=1$ : $\sum_{i \in S} z_i = 2$ , $\sum_{i \in S} p_i = 6$ ✓

$i=2$ : $\sum_{i \in S} z_i = 5$ , $\sum_{i \in S} p_i = 11$ ✓

( $i=3$ : $\sum_{i \in S} z_i = 11$ X )

( $i=4$ : $\sum_{i \in S} z_i = 12$ X )

( $i=5$ : $\sum_{i \in S} z_i = 10$ X )

( $i=6$ : $\sum_{i \in S} z_i = 14$ X )

$i=7$ : $\sum_{i \in S} z_i = 9$ , $\sum_{i \in S} p_i = 14$

Gesamtnutzen 14 ist also erreichbar!

Obere Schranke:

Greedy für das einfachere Problem „Fractional Knapsack", d.h. Algorithmus 1.4.

$i=1$: $\sum_{i \in S} z_i = 2$, $\sum_{i \in S} p_i = 6$

$i=2$: $\sum_{i \in S} z_i = 5$, $\sum p_i = 11$

Bleibt $2 - \sum z_i = 4$, also

$x_3 = \frac{2}{3}$ $\left( = \frac{4}{6} = \frac{2 - \sum z_i}{z_3} \right)$

Damit: $\sum_{i=1}^{3} x_i p_i = 11 + \frac{2}{3} \cdot 8 = 16\frac{1}{3}$.

Wir wissen, dass die Optimallösung ganzzahlig sein muss, also haben wir eine obere Schranke von 16!

(3) Diese Schranken sind beide einfach zu berechnen!

WICHTIG: Für die obere Schranke betrachten wir ein Hilfsproblem, das einfacher ist, weil wir weniger strenge Bedingungen verlangen. So etwas nennt man eine Relaxierung!