Technische Universität Braunschweig

Algorithmen und Datenstrukturen 2

Arne Schmidt

Zusammenfassung

Intro und Greedyalgorithmen

Klausursituation

https://commons.wikimedia.org/wiki/File:IPhO-de-2020R3_theory-exam.jpg

Klausursituation – 30 Minuten später

https://commons.wikimedia.org/wiki/File:IPhO-de-2020R3_theory-exam.jpg

Knapsackprobleme

Problem 1.2 (Rucksackproblem, 0-1-Knapsack).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe $z_i > 0$ Gewinn $p_i > 0$
- Größenschranke Z
- Gewinnschranke P

Gesucht:

Eine Menge

$$S \subseteq \{1, ..., n\}$$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$\sum_{i \in S} p_i \geq P$$

Problem 1.2′ (Maximum Knapsack).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe $z_i > 0$ und Gewinn $p_i > 0$
- Größenschranke Z

Gesucht:

Eine Menge

$$S \subseteq \{1, ..., n\}$$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$\sum_{i \in S} p_i = Maximal$$

Entscheidungsproblem
"Gibt es eine solche Menge mit Wert P überhaupt?"

Optimierungsproblem
"Unter allen gültigen Lösungen, gib mir die beste!"

Fractional Knapsack

Problem 1.3 (FRACTIONAL KNAPSACK).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe $z_i > 0$ und Gewinn $p_i > 0$
- Größenschranke Z

Gesucht:

Für jedes Objekt ein Wert

$$x_i \in [0, 1]$$

sodass

$$\sum_{i=1}^{n} z_i x_i \leq Z$$

und

$$\sum_{i=1}^{n} p_i x_i = Maximal$$

Algorithmus 1.4 Greedy-Algorithmus für FRACTIONAL KNAPSACK

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

Ausgabe: $x_1, ..., x_n \in [0, 1]$

mit

$$\sum_{i=1}^{n} z_i x_i \leq Z$$

und

$$\sum_{i=1}^{n} p_i x_i = Maximal$$

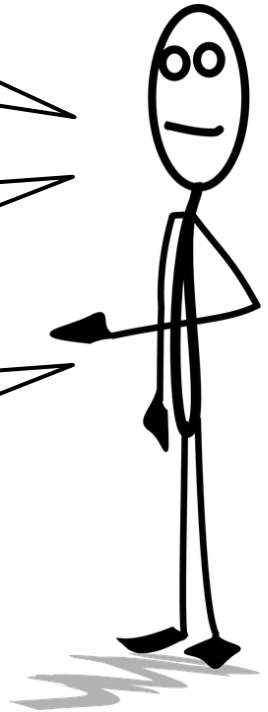
1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(\sum_{i=1}^{j} z_{\pi(i)} \leq Z)$ **do**
3: $\quad x_{\pi(j)} := 1$
4: $\quad j := j + 1$
5: Setze $x_{\pi(j)} := \dfrac{Z - \sum_{i=1}^{j-1} z_{\pi(i)}}{z_{\pi(j)}}$
6: **return**

Greedy-Algorithmen

Greedy-Algorithmen wählen immer die **lokal beste Option**.

Vergangene Entscheidungen spielen keine Rolle mehr.

Sie bieten oft „gute Lösungen“.



Integer Knapsack

Problem 1.6 (INTEGER KNAPSACK).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe z_i Gewinn p_i
- Größenschranke Z

Gesucht:

Für jedes Objekt ein Wert

$$x_i \in \mathbb{N}$$

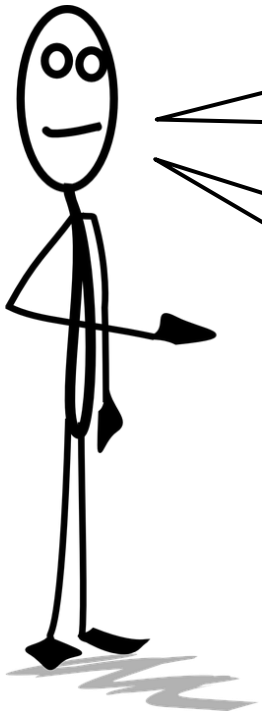
mit

$$\sum_{i=1}^{n} z_i x_i \leq Z$$

und

$$\sum_{i=1}^{n} p_i x_i = Maximal$$

Spezialfall des Spezialfalls: Subsetsum



Alle Objekte besitzen gleiche Dichte

Zielschranke soll exakt erreicht werden.

Problem 1.8 (SUBSET SUM).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe z_i
- Zielgröße Z

Gesucht:

Eine Menge

$$S \subseteq \{1, ..., n\}$$

mit

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

Spezialfall^3 : Partition

Problem 1.9 (PARTITION).

Gegeben:

- n Objekte $1, ..., n$ mit jeweils Größe z_i

Gesucht:

Eine Menge

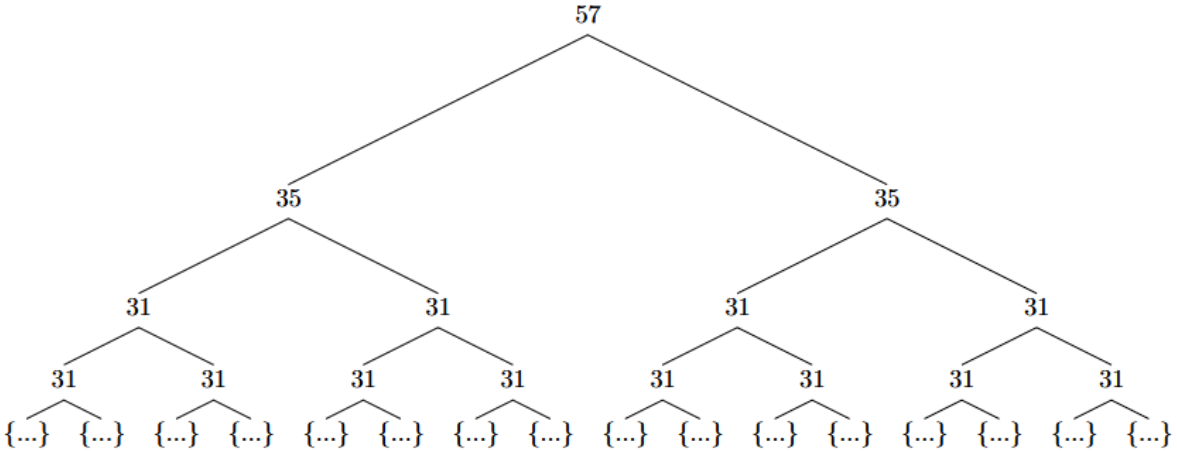
$$S \subseteq \{1, ..., n\}$$

mit

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

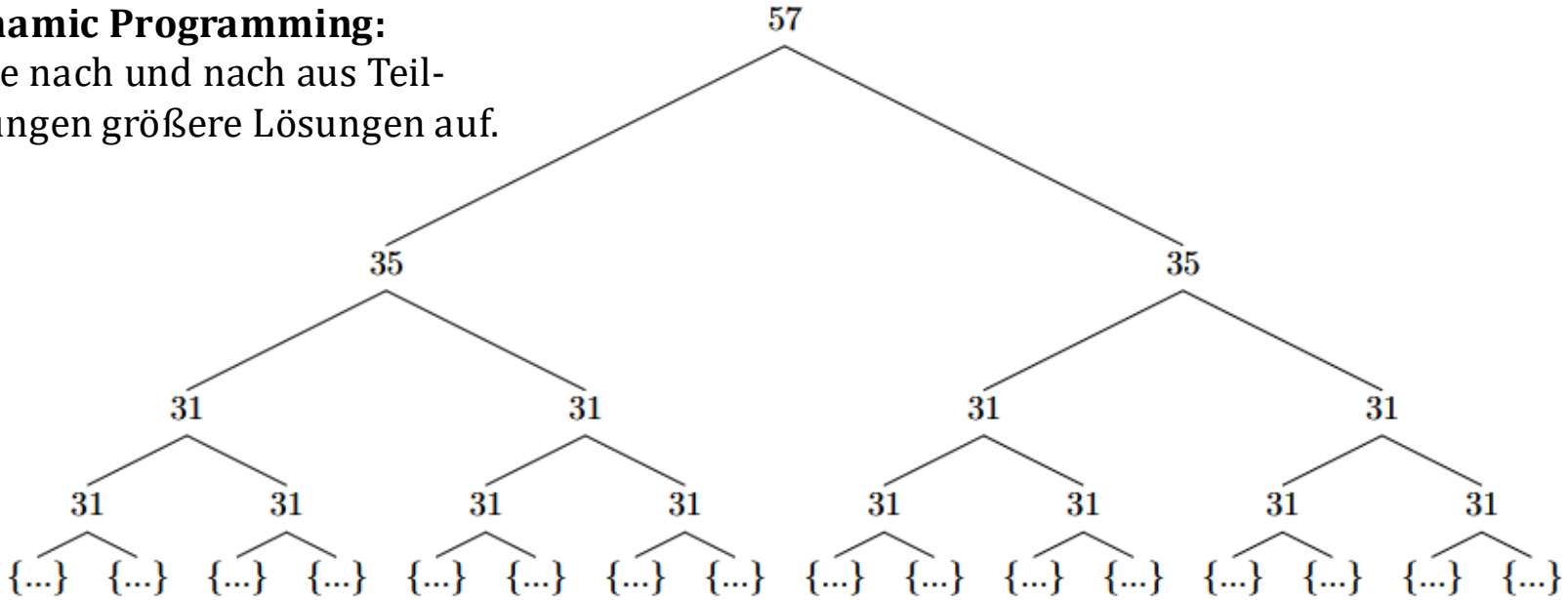
Dynamic Programming vs Branch and Bound

Enumerationsbaum



Enumerationsbaum

Dynamic Programming:
Baue nach und nach aus Teil-lösungen größere Lösungen auf.



Eine Rekursionsgleichung

$$\mathcal{S}(x,i) \coloneqq \begin{cases} 1, & \text{falls } x \text{ mit } z_1, \dots, z_i \text{ erzeugt werden kann.} \\ 0, & \text{sonst} \end{cases}$$

Basisfälle

$$\mathcal{S}(x,0) \coloneqq \begin{cases} 1, & \text{falls } x = 0. \\ 0, & \text{sonst} \end{cases}$$

$$\mathcal{S}(x,i) = \begin{cases} \mathcal{S}(x,i-1), & \text{falls } x < z_i \\ \max\left(\mathcal{S}(x,i-1), \mathcal{S}(x-z_i,i-1)\right), & \text{falls } x \geq z_i. \end{cases}$$

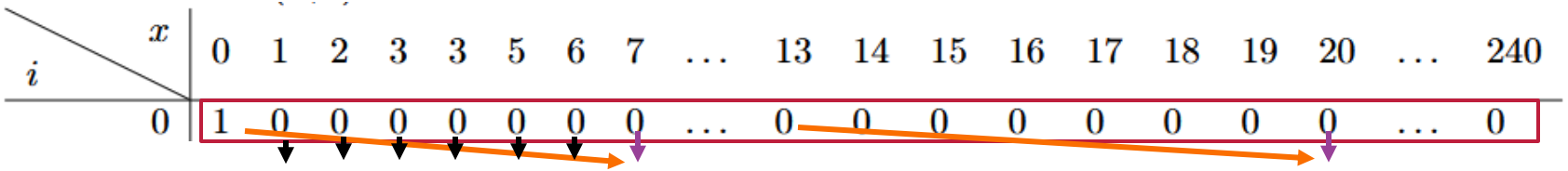
Was passiert, wenn wir es nicht nehmen?

Was passiert, wenn wir es nehmen?

Objekt passt eh nicht.

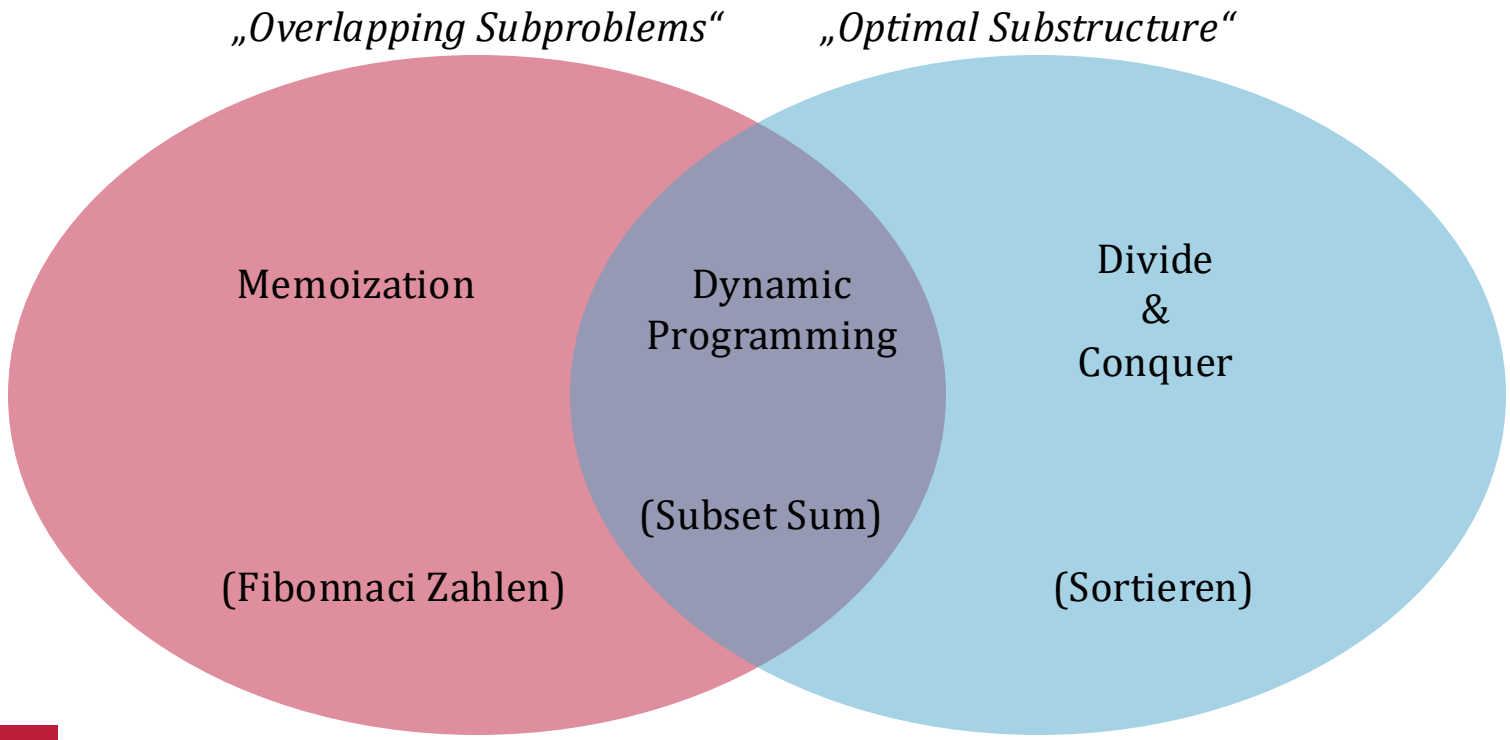
Wertetabelle

$\{z_1, \ldots, z_9\} = \{7, 13, 17, 20, 29, 31, 31, 35, 57\}$



i \ x	0	1	2	3	3	5	6	7	…	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

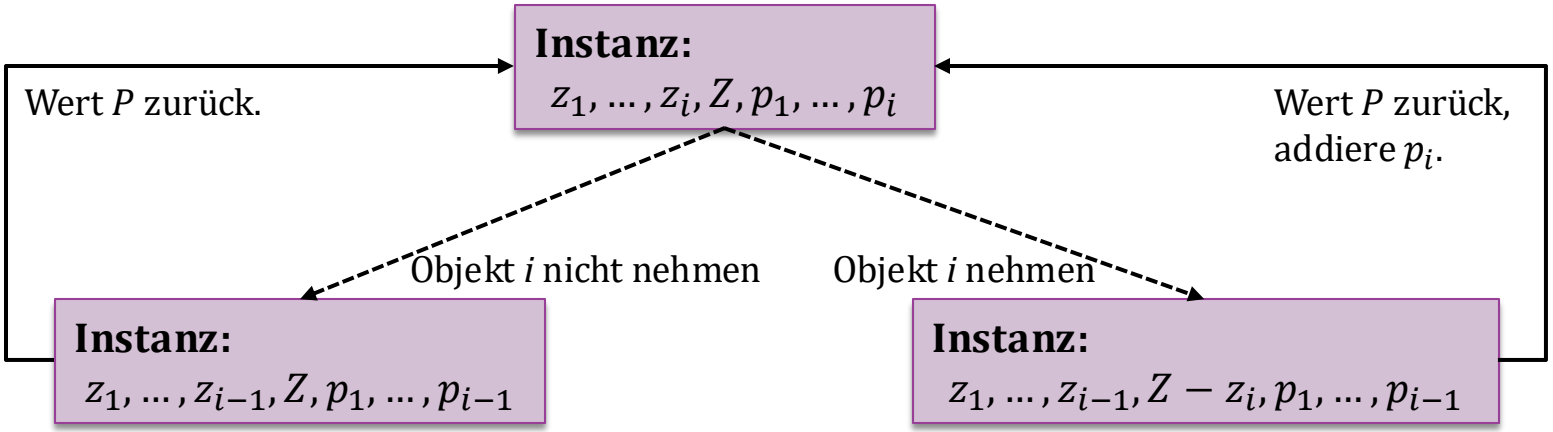
Dynamic Programming



Ansätze für Knapsack

Wir haben nun nicht nur Gewichte, sondern:
z_i und p_i, sowie Z.

Frage:
Gibt es Subprobleme wie bei Subset Sum, deren Lösung uns hilft?



Rekursionsgleichung für Maximum Knapsack

Zu überlegen:
- Worüber geht die Rekursion?
- Was bedeutet die rekursive Funktion?
- Was sind (einfache) Basisfälle?
- Welche Bedingungen gibt es, ein Objekt zu nehmen?

Sei $P(x, i)$ der größtmögliche Wert, der mit den ersten i Objekten mit Kapazität x erreicht werden kann.

Basisfälle

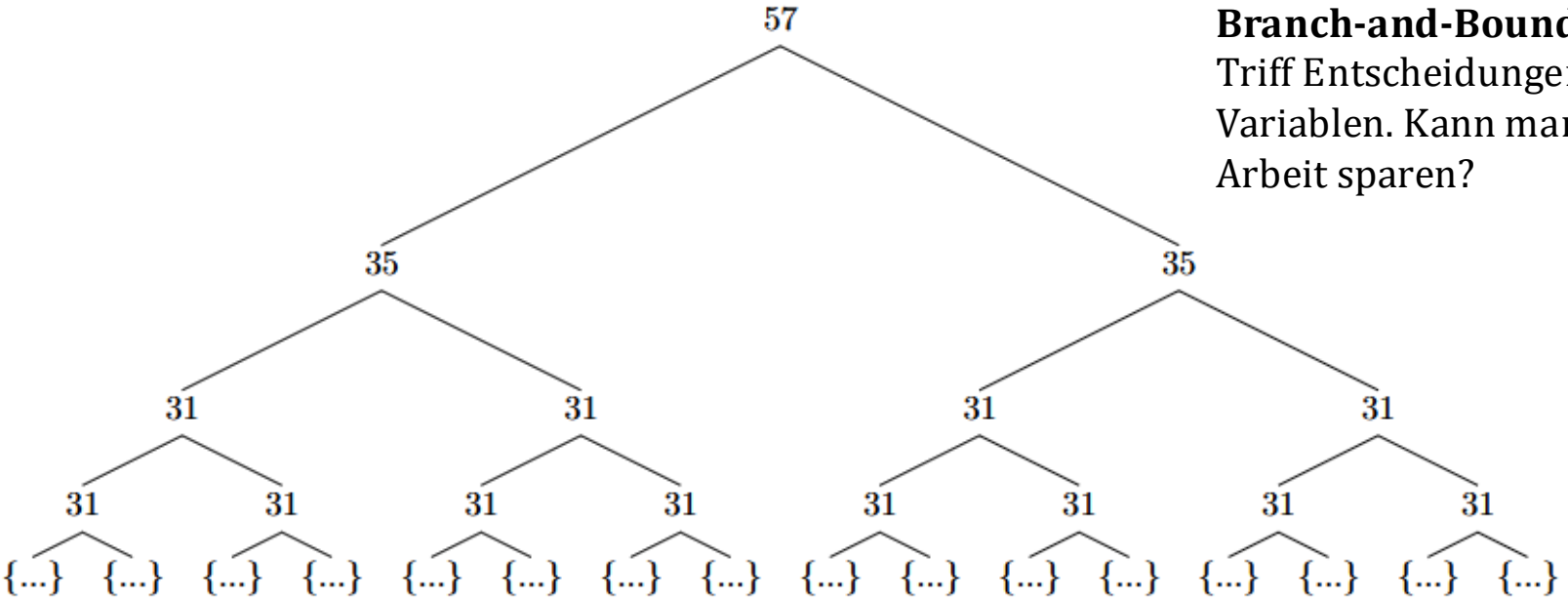
$$P(x,i) = \begin{cases} 0, & \text{falls } i = 0 \\ P(x, i-1), & \text{falls } x < z_i \\ \max(P(x,i-1), P(x-z_i, i-1) + p_i), & \text{falls } x \geq z_i. \end{cases}$$

Objekt passt eh nicht.

Was passiert, wenn wir es nicht nehmen?

Was passiert, wenn wir es nehmen?

Enumerationsbaum



Branch-and-Bound:
Triff Entscheidungen für Variablen. Kann man hier Arbeit sparen?

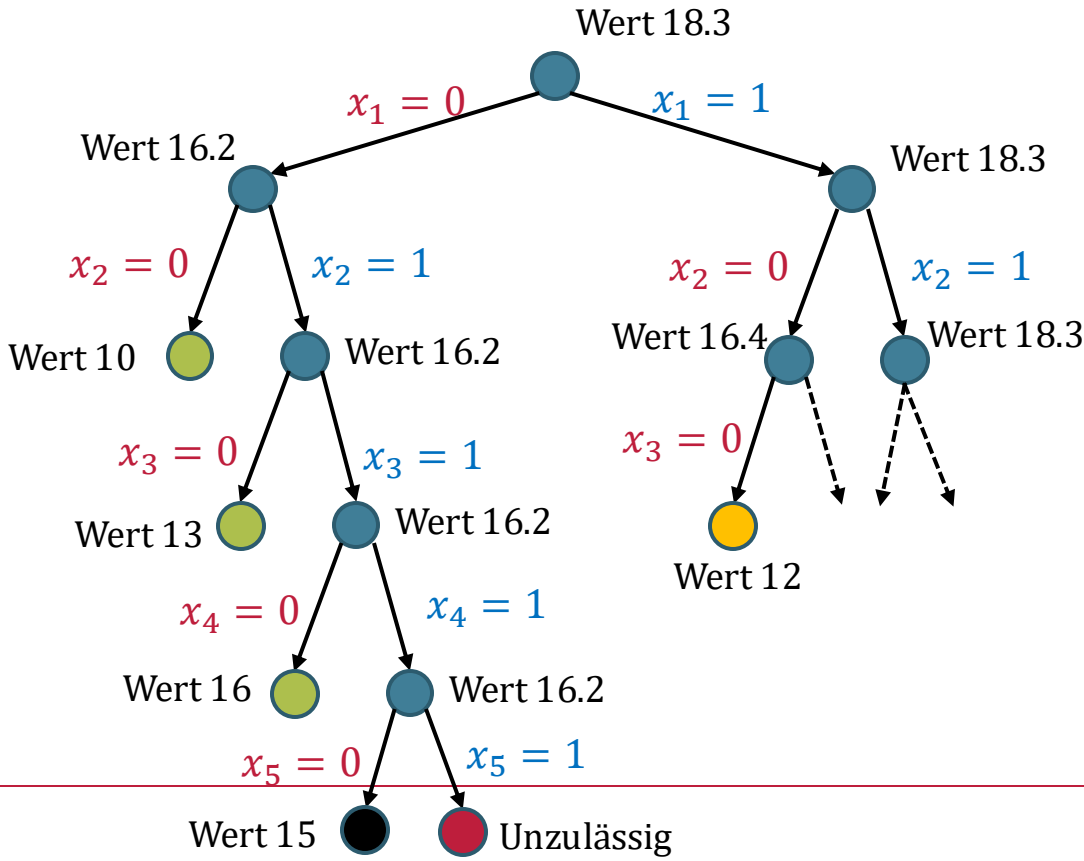
Beispiel: Maximum Knapsack

Zusammenfassend:

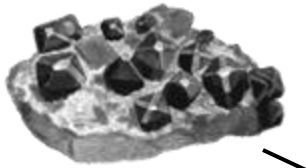
- **(Grüner Knoten):** Bessere Ganzzahlige Lösung gefunden!
- **(Gelber Knoten)** Es gibt hier keine bessere Lösung als bisher!
- **(Roter Knoten)** Unzulässige Entscheidungen!
- **(Schwarzer Knoten)** Wir können keine Entscheidung mehr treffen!

Dabei wichtig:

- Die obere Schranke ist nur für den Teilbaum gültig.
- Die Untere (zulässige Lösung) für den gesamten Enumerationsbaum!



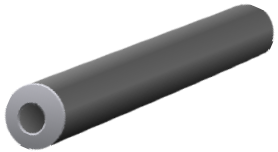
Lineare Programme



Eisen

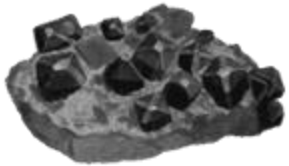
240t pro Monat

Stahlrohre (1000€ / t)



0.8t Eisen + 0.4t Kohle pro t

0.9t Eisen + 0.7t Kohle pro t



Kohle

180t pro Monat

Stahlträger (1400€ / t)

Etwas anders

Maximiere $(1000, 1400) \cdot \begin{pmatrix} Rohre \\ Träger \end{pmatrix}$ **Profit**

$$\begin{pmatrix} 0.8 & 0.9 \\ 0.4 & 0.7 \end{pmatrix} \begin{pmatrix} Rohre \\ Träger \end{pmatrix} \leq \begin{pmatrix} 240 \\ 180 \end{pmatrix}$$

Eisenlimit

$$\begin{pmatrix} Rohre \\ Träger \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Kohlelimit

Allgemein

Maximiere $c^T x$

Unter Bedingungen

$$Ax \leq b$$

$$x \geq 0$$

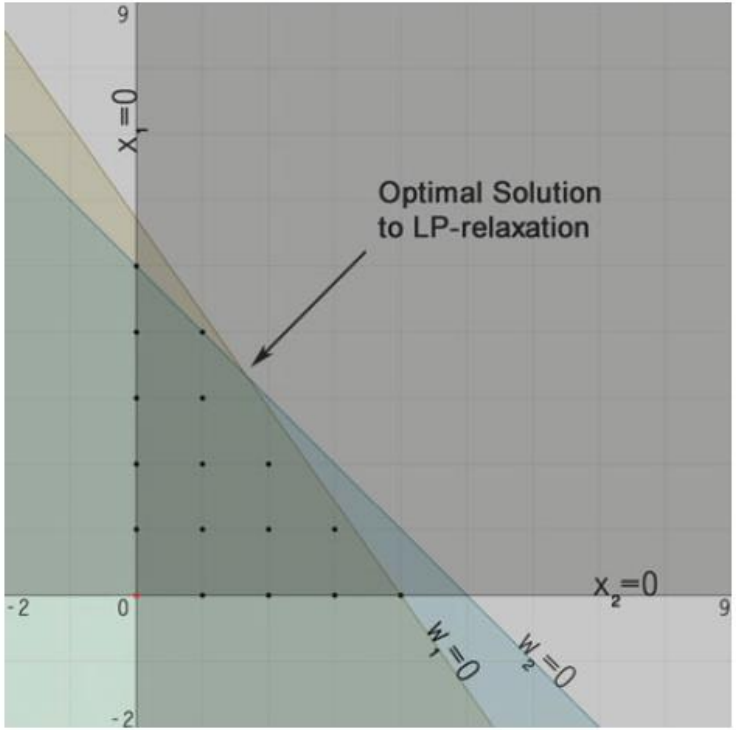
Beispiel (1)

$$
\begin{aligned}
\text{maximize } & 17x_1 + 12x_2 \\
\text{subject to } & 10x_1 + 7x_2 \leq 40 \\
& x_1 + x_2 \leq 5 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ integers}
\end{aligned}
$$

Optimale Lösung der Relaxierung:

$$x = \left(\frac{5}{3}, \frac{10}{3}\right)$$

Betrachte Subprobleme mit
$x_1 \leq 1$ und $x_1 \geq 2$



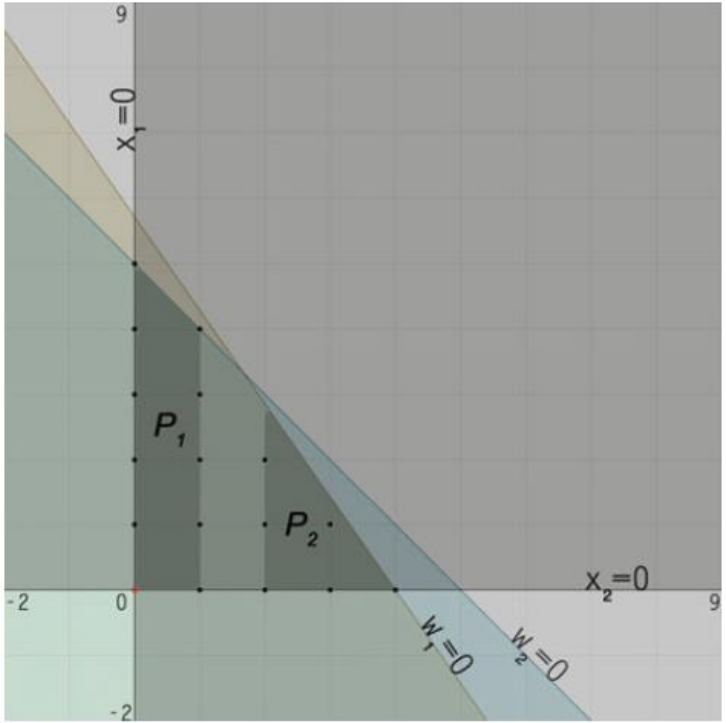
Beispiel (2)

$$
\begin{aligned}
\text{maximize } & 17x_1 + 12x_2 \\
\text{subject to } & 10x_1 + 7x_2 \leq 40 \\
& x_1 + x_2 \leq 5 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \text{ integers}
\end{aligned}
$$

Optimale Lösung der Relaxierung:

$$x = \left(\frac{5}{3}, \frac{10}{3}\right)$$

Betrachte Subprobleme mit
$x_1 \leq 1$ und $x_1 \geq 2$

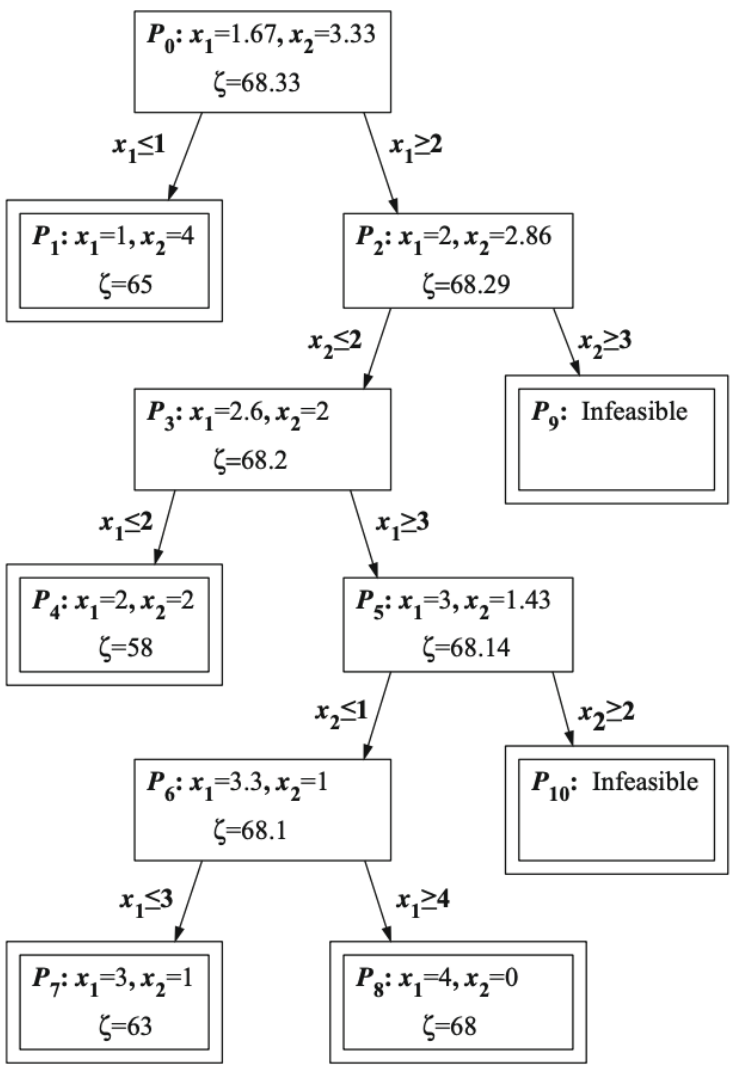


Beispiel (n)

$$
\begin{aligned}
\text{maximize} \quad & 17x_1 + 12x_2 \\
\text{subject to} \quad & 10x_1 + 7x_2 \leq 40 \\
& x_1 + x_2 \leq 5 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ integers}
\end{aligned}
$$

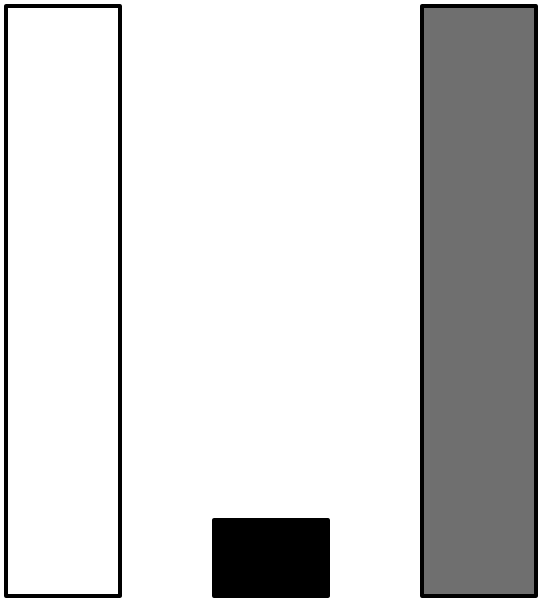
Wir bauen uns nach und nach einen **Enumerationsbaum** auf.

Die beste ganzzahlige Lösung ist also $x = (4,0)$.



Approximationsalgorithmen

Greedy beliebig schlecht



$Z = N$ $\quad z_1 = 1$ $\quad z_2 = N$

$n = 2$ $\quad p_1 = 2$ $\quad p_2 = N$

Algorithmus 4.2 GREEDY$_0$

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

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Inklusionsmaximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: **return**

Triviale Verbesserung

Algorithmus 4.3 Greedy-Approximation

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

Ausgabe: $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G'_0 := \sum_{i \in S} p_i$

1: $G'_0 := max\{G_0, max\{p_i \mid z_i \leq Z, i \in \{1, ..., n\}\}\}$

2: **return**

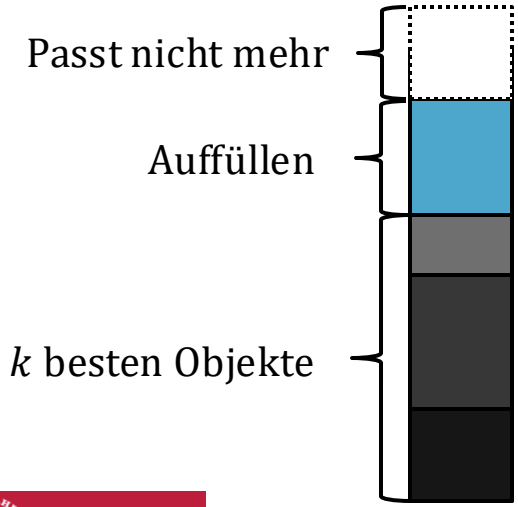
Nimm das Maximum von…

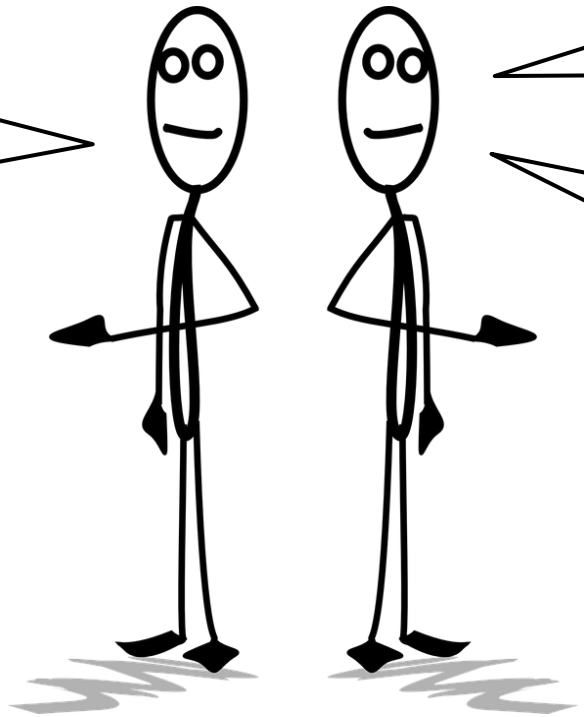
Greedy0

Wertvollstem Objekt

Teilmengen enumerieren

Benutze die k besten Objekt und fülle sie auf. Der Fehler ist durch höchstens 1 Objekt beschränkt!





Enumeriere alle Teilmengen der Größe höchstens $k \in \mathbb{N}$.

Das sind $O(n^k)$ viele.
Für fixes k ist das polynomiell!

Greedy-k

Algorithmus 4.6 $\textsc{Greedy}_k$

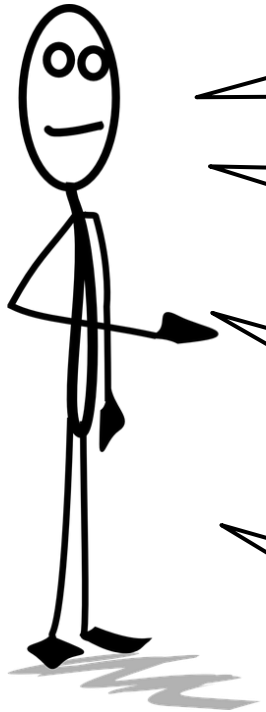
Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$ [Parameter k fixiert]
Ausgabe: $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G_k := \sum_{i \in S} p_i$

1: $G_k := 0$, $S := \emptyset$
2: **for all** $\bar{S} \subseteq \{1, ..., n\}$ mit $|\bar{S}| \leq k$ **do**
3: **if** $(\sum_{i \in \bar{S}} z_i \leq Z)$ **then**
4: $G_k := max\{G_k, \sum_{i \in \bar{S}} p_i + \textsc{Greedy}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\}$;
5: Update S;
6: **return** G_k, S

Satz 4.7:
$\textsc{Greedy}_k$ ist $\left(1 - \frac{1}{k+1}\right)$-Approximationsalgorithmus.

Komplexität

Laufzeiten zum Lösen von Maximum Knapsack



Greedy $O(n \log n)$
Vorteil: Liefert schnell optimale Lösungen
Nachteil: Optimale Lösungen sind nicht immer da…

Dyn. Programming $O(nZ)$
Vorteil: Liefert immer optimale Lösungen
Nachteil: Langsam!

Branch-and-Bound $O(n2^n)$
Vorteil: Liefert immer optimale Lösungen
Nachteil: Langsam!

Greedy-k $O(n^{k+2})$
Vorteil: Liefert immer gute Lösungen
Nachteil: selten optimal!

Die Klassen P und NP

Definition 5.1
Ein algorithmisches Problem Π gehört zur Klasse P, wenn es für Π einen Algorithmus mit polynomieller Laufzeit (in der Inputgröße) gibt, der immer eine optimale Lösung findet.

Definition 5.2
Ein Problem Π gehört zur Klasse NP, wenn für jede Instanz I von Π die Gültigkeit einer Lösung in polynomieller Zeit (in der Inputgröße von I) nachgeprüft werden kann.

Bemerkung
In der Regel werden die Klassen P und NP über Entscheidungsprobleme definiert: “Gibt es eine Lösung?”
Bspw ist 0-1 Knapsack ein Entscheidungproblem.

Eine formalere Definition gibt es bspw. in Theoretische Informatik 2

3SAT auf Knapsack

1. Ziffer: Man muss Objekt 1 oder 2 wählen, aber nicht beide.

2. Ziffer: Man muss Objekt 3 oder 4 wählen, aber nicht beide.

3. Ziffer: Man muss Objekt 5 oder 6 wählen, aber nicht beide.

4. Ziffer: Man muss Objekt 1, 3 oder 6 wählen.

5. Ziffer: Man muss Objekt 1, 4 oder 6 wählen.

6. Ziffer: Man muss Objekt 2, 3 oder 5 wählen.

$\rightarrow (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$

$n = 12,\ z_i = p_i,\ Z = 111444$

$$
\begin{aligned}
z_1 = p_1 &= 100110 \\
z_2 = p_2 &= 100001 \\
z_3 = p_3 &= 10101 \\
z_4 = p_4 &= 10010 \\
z_5 = p_5 &= 1001 \\
z_6 = p_6 &= 1110 \\
z_7 = p_7 &= 200 \\
z_8 = p_8 &= 100 \\
z_9 = p_9 &= 20 \\
z_{10} = p_{10} &= 10 \\
z_{11} = p_{11} &= 2 \\
z_{12} = p_{12} &= 1
\end{aligned}
$$

NP-schwer und -Vollständigkeit

Definition 5.13

1. Ein Problem Π heißt NP-schwer, wenn eine Polynomialzeitreduktion von jedem NP-Problem auf Π existiert.
2. Ein Problem Π heißt NP-vollständig, wenn $\Pi \in NP$ und Π NP-schwer ist.

Bemerkung:
Für den ersten Teil genügt es, ein bereits bekanntes NP-schweres Problem auf Π zu reduzieren!

Korollar 5.14

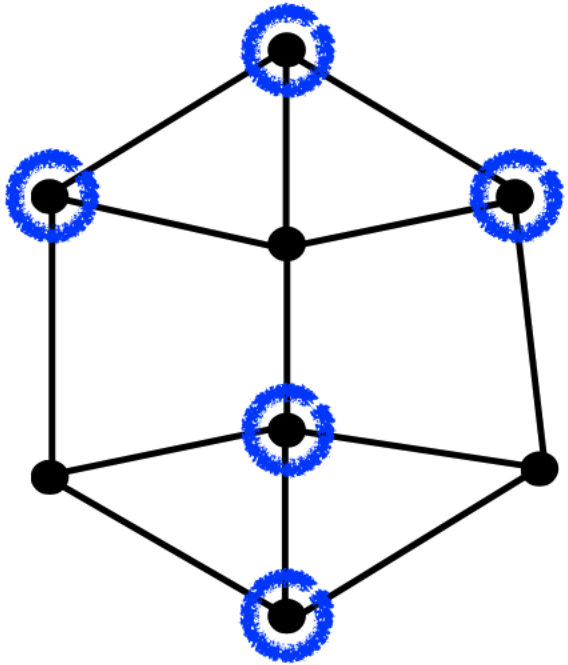
1. 3SAT ist NP-vollständig
2. 0-1 Knapsack ist NP-vollständig
3. Maximum Knapsack ist NP-schwer

Frage: Warum ist nicht klar, ob Maximum Knapsack in NP liegt?

Vertex Cover

Problem Vertex Cover (Optimierungsvariante)
Gegeben:
- Ein Graph $G = (V, E)$

Gesucht:
- Knotenmenge $S \subseteq V$ möglichst kleiner Kardinalität, sodass $u \in S$ oder $v \in S$ für jedes $e = \{u, v\} \in E$ gilt.



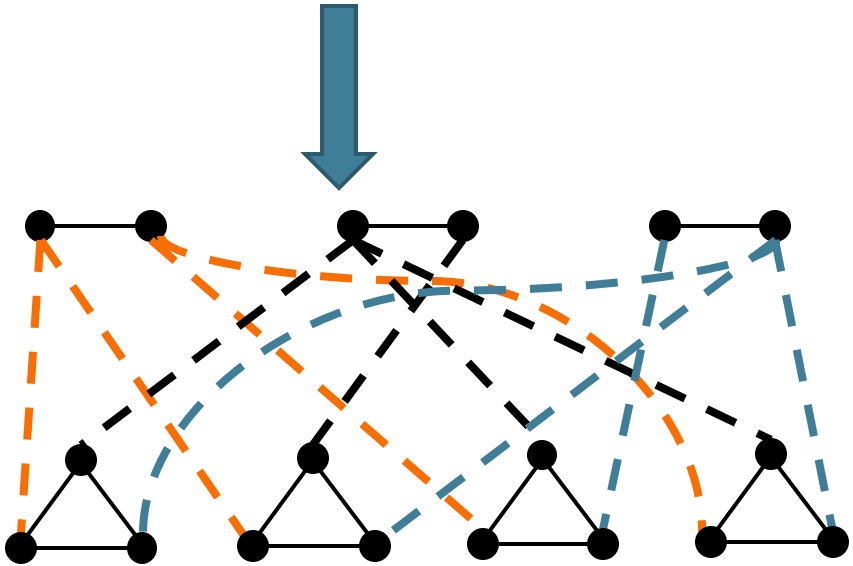
Für die Entscheidungsvariante ist zusätzlich ein $k \in \mathbb{N}$ gegeben, und man fragt, ob es ein Vertex Cover der Größe höchstens k gibt.

Eine Lösung können wir auch schnell überprüfen!
→ Vertex Cover ist in NP.

Reduktion 3SAT auf Vertex Cover

$\varphi := (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

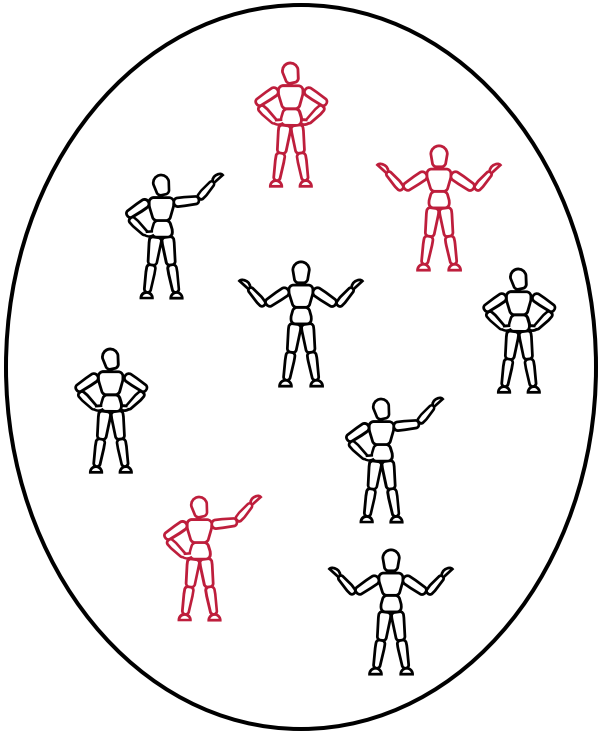
G



φ ist genau dann erfüllbar, wenn G ein Vertex Cover der Größe 11 besitzt.

Hashing

Großes Universum, kleine Welt



Sehr große Anzahl an möglichen Datensätzen.

Am Ende wird nur ein ganz kleiner Teil benötigt!

Was passiert mit diesen Datensätzen?

Aus AuD 1, dynamische Verwaltung der Daten:

- Search(x), insert(x), delete(x)
- Datenstrukturen:
  - Binäre Suchbäume
  - Heaps
  - ...

Hashtabelle

Definition 7.1
Eine **Hashtabelle** der Größe m besteht aus einem Array T mit Speicherzellen $T[0], \dots, T[m-1]$. Die Datensätze werden in dieser Hashtabelle gespeichert.

Eine **Hashfunktion** $h: U \to \{0, \dots, m-1\}$ liefert für jeden Schlüssel x aus einer Menge U eine Adresse in der Hashtabelle

Für gut gewähltes m und h können Operationen (search, insert, delete) in erwarteter konstanter Zeit durchgeführt werden.

Datenstrukturen aus AuD 1 benötigen im Mittel $O(\log n)$ Zeit.

Definition 7.2
Der Quotient $\beta = \frac{n}{m}$ einer Hashtabelle mit m Speicherzellen und n eingefügten Datensätzen heißt **Belegungsfaktor**.

Hashverfahren

Definition 7.3
Ein Hashverfahren ist gegeben durch:
- Eine Hashtabelle
- Eine Hashfunktion
- Strategie zur Auflösung möglicher Adresskollisionen.

Die Hashfunktion sollte dabei
- **Surjektiv** sein (ganzer Wertebereich wird abgedeckt)
- Schlüssel möglichst gleichmäßig verteilen, d.h. für $i, j \in \{0, \dots, m-1\}$ ist $|h^{-1}(i)| \approx |h^{-1}(j)|$
- Effizient berechenbar sein

Wir nehmen vereinfacht an, dass $U = \{0, 1, \dots, N-1\}$.
Eine einfache Hashfunktion ist dann $h(x) = x \bmod m$.

Geburtstagsparadoxon

Wie wahrscheinlich ist es, dass zwei Personen von n zufällig gewählten am gleichen Tag Geburtstag haben?

Wie viele Personen benötigt man für eine Chance von ~50%?

JANUARY

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

FEBRUARY

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

MARCH

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

APRIL

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

MAY

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNE

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

JULY

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

AUGUST

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

SEPTEMBER

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

OCTOBER

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

NOVEMBER

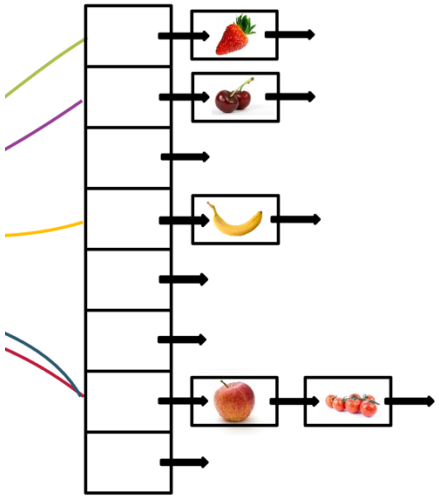
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

DECEMBER

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

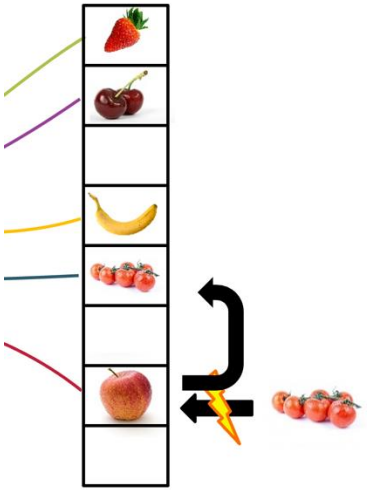
Kollisionsbehandlung

Listen



Daten mit gleichem Hashwert werden in einer verketteten Liste gespeichert.

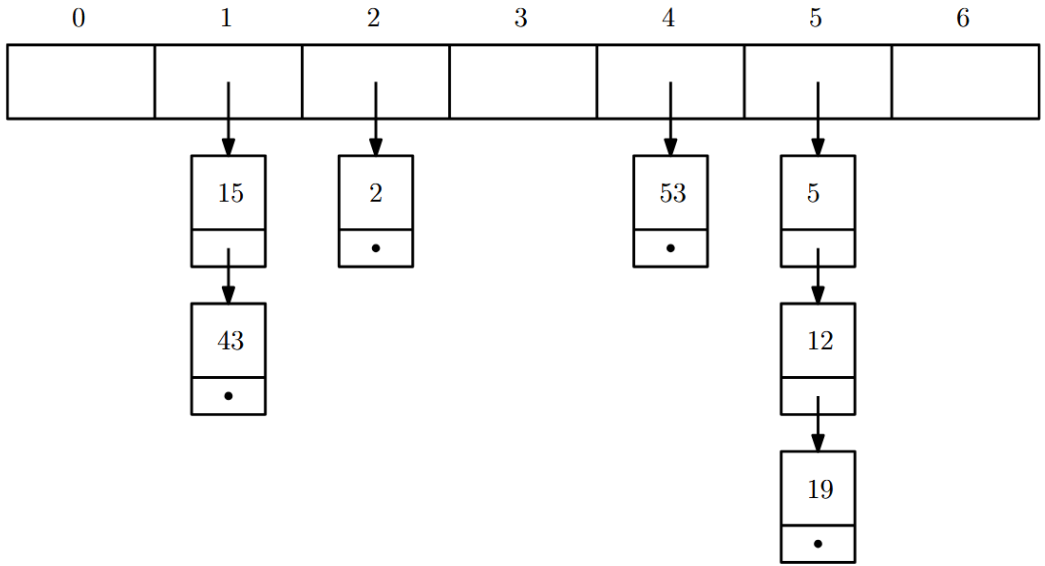
Offene Adressierung



Suche nach fester Regel (Sondierungsregel) einen freien Platz

Beispiel 7.8

Sei $m = 7$ und $h(x) = x \bmod m$.
Betrachte Schlüsselmenge $S = \{2, 5, 12, 15, 19, 43, 53\}$. Einfügen in die Hashtabelle liefert



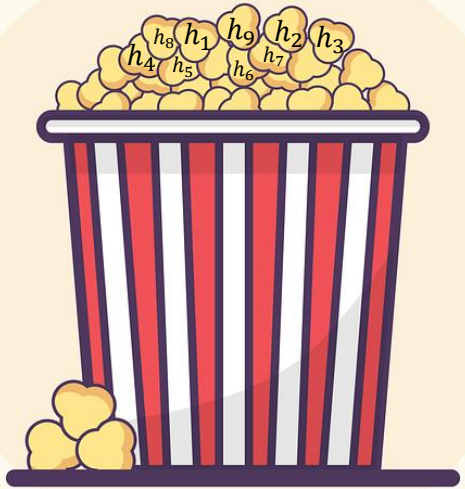
Doppeltes Hashing

Nutze zwei Hashfunktionen $h_1(x), h_2(x)$, welche additiv verknüpft werden.
Einer der beiden wird multiplikativ mit dem Versuchszähler $0 \leq i \leq m-1$ verknüpft.

Beispiel 7.16
Sei $m = 19, x = 47$ und $h(i,x) = h_1(x) + i \cdot h_2(x) \bmod 19$ mit $h_1(47) = 47 \bmod 19 = 9$ und $h_2(47) = 47 \bmod 17 + 1 = 14$. Dann ist die Sondierungsfolge:
9, 4, 18, 13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 0, 14

Beobachtung 7.17
$i \cdot h_2(x)$ durchläuft für $1 \leq i \leq m-1$ die Werte $1, \ldots, m-1$ in irgendeiner Reihenfolge.
Ergänzt wird $0 = 0 \cdot h_2(x)$.
Durch den Summanden $h_1(x)$ wird der Start zufällig verschoben.
Doppeltes Hashing kommt dem idealen Hashing am nächsten.

Universelles Hashing



Wähle zur Laufzeit eine Hashfunktion aus einer Familie von Hashfunktionen mit besonderen Eigenschaften.

Damit hat man für jede Schlüsselmenge eine zufällige Hashfunktion und im Erwartungsfall eine gute Laufzeit.

Das Ende!

Nächste Woche: Fragestunde!

Eure Fragen bitte bis
Montag, 14.07., 12 Uhr
per Mail aschmidt@ibr.cs.tu-bs.de