Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

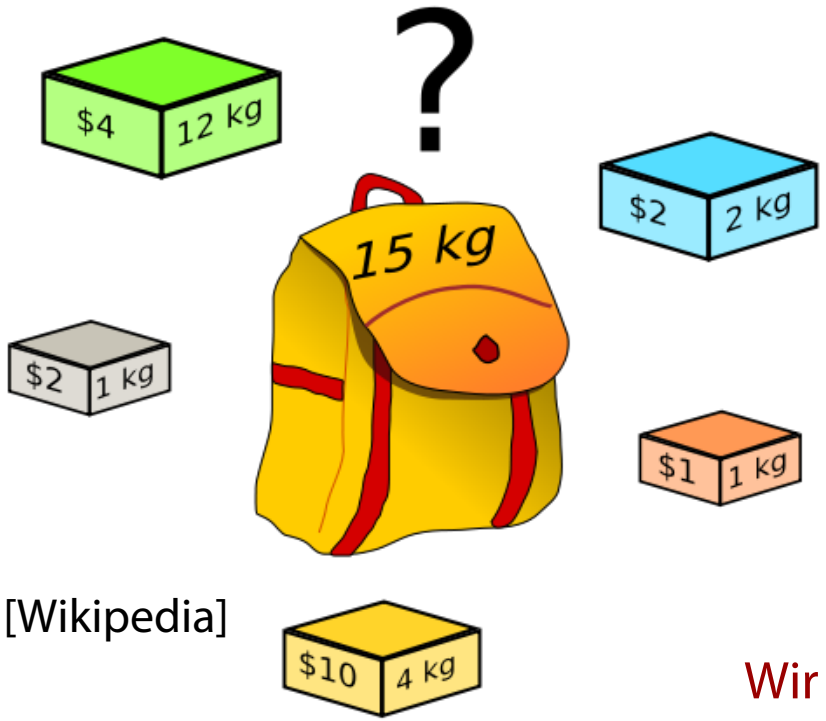
Felix Kuhr (Übungen)

sowie viele Tutoren

Rucksackproblem

Gegeben sei eine Menge von Gegenständen

- Jeder Gegenstand hat einen Wert und ein Gewicht
- Ziel: Maximiere Wert der Dinge im Rucksack
- Einschränkung: Rucksack hat Gewichtsgrenze



[Wikipedia]

Drei Versionen:

0-1-Rucksackproblem: Wähle Gegenstand oder nicht

Gestückeltes Rucksackproblem: Gegenstände sind teilbar

Unbegrenztes Rucksackproblem: Beliebige Anzahlen verfügbar

Welches ist das leichteste Problem?

Wir beginnen mit dem 0-1-Problem

0-1-Problem

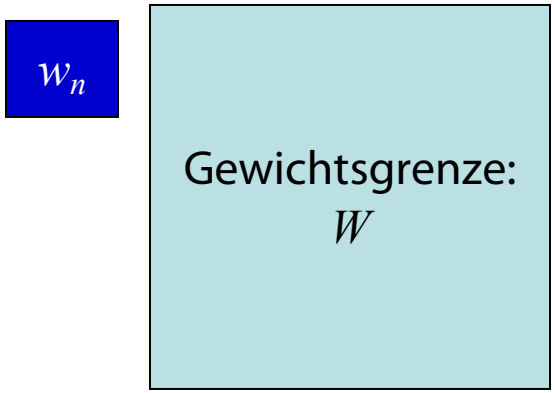
- Rucksack hat Gewichtseinschränkung W
- Gegenstände $1, 2, \ldots, n$ (beliebig)
- Gegenstände haben Gewichte $w_1, w_2, \ldots, w_n$
  - Gewichte sind Ganzzahlen und $w_i < W$
- Gegenstände haben Werte $v_1, v_2, \ldots, v_n$
- Ziel: Finde eine Teilmenge $S \subseteq \{1, 2, ..., n\}$ von Gegenständen, so dass $\sum_{i \in S} w_i \leq W$ und $\sum_{i \in S} v_i$ maximal bezüglich aller möglicher Teilmengen

Naive Algorithmen

- Betrachte alle Untermengen $S \subseteq \{1, 2, ..., n\}$
  - Optimale Lösung wird gefunden, aber exponentiell
- Gierig 1: Nimm jeweils den nächsten passenden Gegenstand mit dem größten Wert
  - Optimale Lösung wird nicht gefunden
  - Wer kann ein Beispiel geben?
- Gierig 2: Nehme jeweils nächsten Gegenstand mit dem besten Wert/Gewichts-Verhältnis
  - Optimale Lösung wird nicht gefunden
  - Wer kann ein Beispiel geben?

Ansatz mit dynamischer Programmierung

- Angenommen, die optimale Lösung S ist bekannt
- Fall 1: Gegenstand n ist im Rucksack
- Fall 2: Gegenstand n ist nicht im Rucksack



Finde optimale Lösung unter Verwendung von Gegenständen 1, 2, …, n-1 mit Gewichtsgrenze $W - w_n$

Finde optimale Lösung unter Verwendung von Gegenständen 1, 2, …, n-1 mit Gewichtsgrenze W

Rekursive Formulierung

Sei V[i, w] der optimale Gesamtwert wenn Gegenstände 1, 2, …, i betrachtet werden für aktuelle Gewichtsgrenze w

=> V[n, W] ist die optimale Lösung

$$V[n, W] = \max \begin{cases} V[n-1, W-w_n] + v_n \\ V[n-1, W] \end{cases}$$

↓ Generalisierung

$$V[i, w] = \begin{cases} \max \begin{cases} V[i-1, w-w_i] + v_i & \text{Ggst i gewählt} \\ V[i-1, w] & \text{Ggst i nicht gewählt} \end{cases} \\ V[i-1, w] \text{ if } w_i > w & \text{Ggst i nicht gewählt} \end{cases}$$

Grenzfall: V[i, 0] = 0, V[0, w] = 0. Wieviele Teilprobleme?

Beispiel

- n = 6 (# der Gegenstände)
- W = 10 (Gewichtsgrenze)
- Gegenstände (Gewicht und Wert):

	w_i	v_i
1	2	2
2	4	3
3	3	3
4	5	6
5	2	4
6	6	9

	w_i	v_i
1	2	2
2	4	3
3	3	3
4	5	6
5	2	4
6	6	9

			w	0	1	2	3	4	5	6	7	8	9	10
i	w_i	v_i												
				0	0	0	0	0	0	0	0	0	0	0
1	2	2		0										
2	4	3		0				w_i						
3	3	3		0		V[i-1,	w-w_i]				V[i-1,	w]		
4	5	6		0							V[i, w]			
5	2	4		0										
6	6	9		0										

$$V[i, w] = \begin{cases} \max \begin{cases} V[i-1, w-w_i] + v_i & \text{falls Gegenstand } i \text{ verwendet} \\ V[i-1, w] & \text{falls Gegenstand } i \text{ nicht verwendet} \end{cases} \\ V[i-1, w] \quad \text{falls } w_i > w \quad \text{Ggst. } i \text{ nicht verwendet} \end{cases}$$

			w	0	1	2	3	4	5	6	7	8	9	10
i	w_i	v_i												
				0	0	0	0	0	0	0	0	0	0	0
1	2	(2)		0	0	2	2	2	2	2	2	2	2	2
2	4	3		0	0	2	2	3	3	5	5	5	5	5
3	3	(3)		0	0	2	3	3	5	5	6	6	8	8
4	5	6		0	0	2	3	3	6	6	8	9	9	11
5	2	(4)		0	0	4	4	6	7	7	10	10	12	13
6	6	9		0	0	4	4	6	7	9	10	13	13	15

$$V[i, w] = \begin{cases} \max \begin{cases} V[i-1, w-w_i] + v_i & \text{falls Gegenstand } i \text{ verwendet} \\ V[i-1, w] & \text{falls Gegenstand } i \text{ nicht verwendet} \end{cases} \\ V[i-1, w] \quad \text{falls } w_i > w \quad \text{Ggst. } i \text{ nicht verwendet} \end{cases}$$

			w	0	1	2	3	4	5	6	7	8	9	10
i	w_i	v_i												
				0	0	0	0	0	0	0	0	0	0	0
1	2	2		0	0	2	2	2	2	2	2	2	2	2
2	4	3		0	0	2	2	3	3	5	5	5	5	5
3	3	3		0	0	2	3	3	5	5	6	6	8	8
4	5	6		0	0	2	3	3	6	6	8	9	9	11
5	2	4		0	0	4	4	6	7	7	10	10	12	13
6	6	9		0	0	4	4	6	7	9	10	13	13	15

Optimaler Wert: 15

Gegenstand: 6, 5, 1

Gewichte: 6 + 2 + 2 = 10

Wert: 9 + 4 + 2 = 15

Analyse

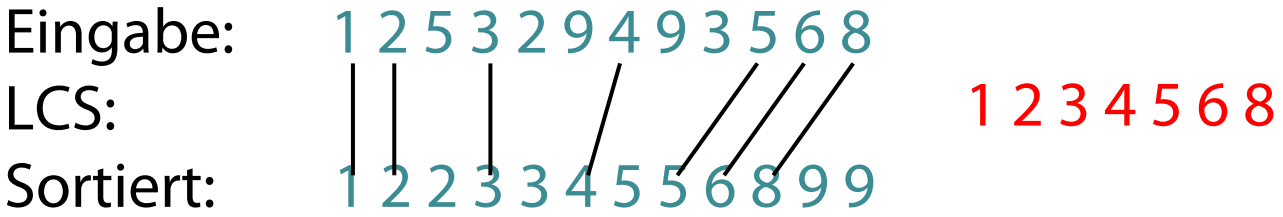
- $\Theta(nW)$
- Polynomiell?
  - Pseudo-polynomiell
    - Laufzeit hängt vom numerischen Wert ab
    - Numerische Werte von der Länge klein (log w)
    - Verdopplung der Länge heißt dann aber exponentiell höhere Werte
  - Funktioniert trotzdem gut, wenn W klein
- Betrachte folgende Gegenstände (Gewicht, Wert):

  (10, 5), (15, 6), (20, 5), (18, 6)
- Gewichtsgrenze 35
  - Optimale Lösung: Ggst. 2, 4 (Wert = 12). Iterationen: $2^4 = 16$ Mengen
  - Dynamische Programmierung: Fülle Tabelle 4 x 35 = 140 Einträge
- Was ist das Problem?
  - Viele Einträge nicht verwendet: Gewichtskombination nicht möglich
  - Brute-Force-Ansatz kann besser sein

Longest-Increasing-Subsequence-Problem

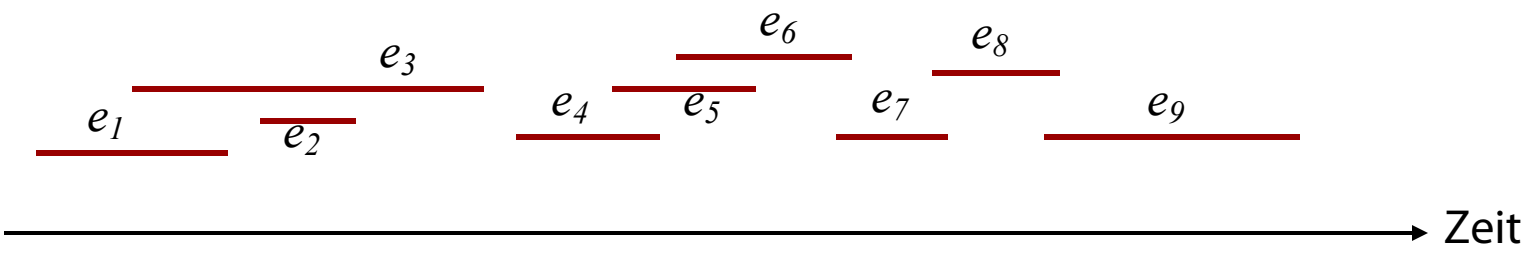
- Gegeben sei eine Liste von Zahlen

  1 2 5 3 2 9 4 9 3 5 6 8
- Finde ***längste*** Teilsequenz, in der keine Zahl kleiner ist als die vorige
  - Beispiel: 1 2 5 9
  - Teilsequenz der originalen Liste, aber nicht die längste
  - Die Lösung ist Teilsequenz der sortierten Liste



- Beispiel zeigt Rückführung auf bekanntes Problem mit Vorverarbeitung der Originaleingabe in O(n log n) bei kleinen Zahlen in O(n)

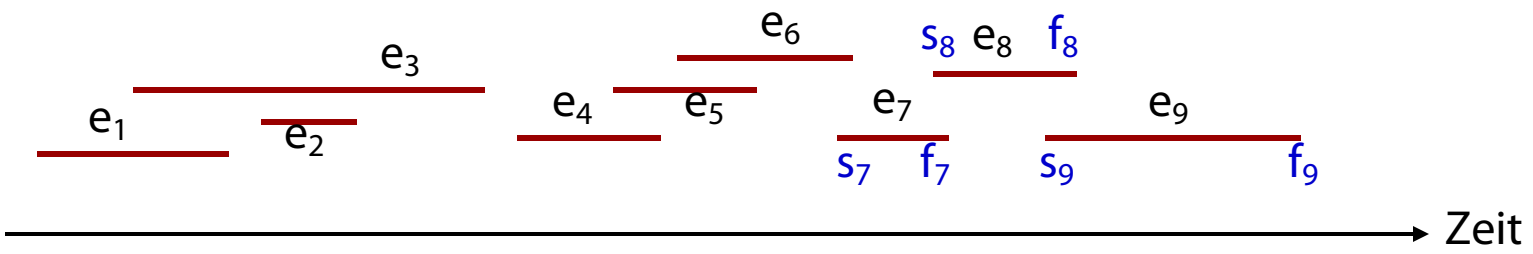
Ereignisplanungs-Problem



- Eingabe: Ein Liste von Ereignissen (Intervallen) zur Berücksichtigung
  - e_i hat Anfangszeit s_i und Endezeit f_i
  - wobei gilt, dass $f_i < f_j$ falls $i < j$
- Jedes Ereignis hat einen Wert v_i
- Gesucht: Plan, so dass Gesamtwert maximiert
  - Nur ein Ereignis kann pro Zeitpunkt stattfinden
- Vorgehen ähnlich zur Restaurant-Platzierung
  - Sortiere Ereignisse nach Endezeit
  - Betrachte ob letztes Ereignis enthalten ist oder nicht

Ereignisplanungs-Problem

Rückführung auf Optimierung durch Dynamische Programmierung



- V(i) ist der optimale Wert, der erreicht werden kann, wenn die ersten i Ereignisse betrachtet werden

- $V(n) = \max \{ \begin{array}{ll} V(n-1) & e_n \text{ nicht gewählt} \\ V(j) + v_n & e_n \text{ gewählt} \end{array}$

$j < n$, j maximal und $f_j < s_n$

Münzwechselproblem

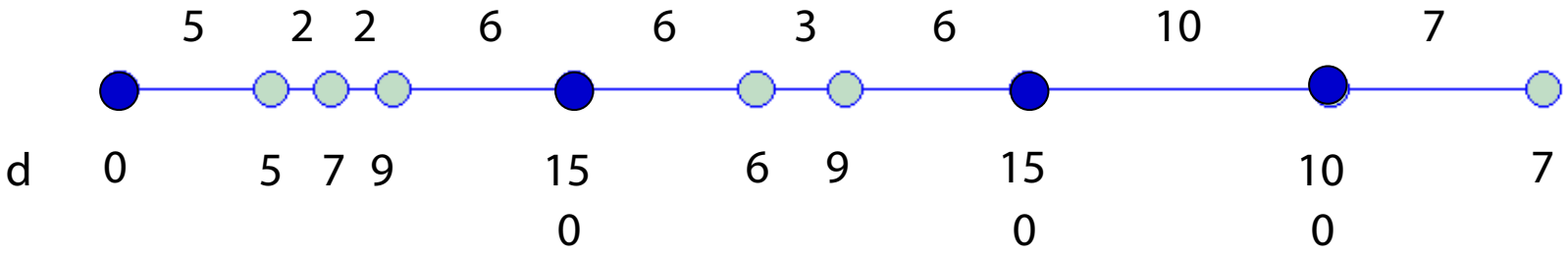
- Gegeben eine Menge von Münzwerten (z.B. 2, 5, 7, 10), entscheide, ob es möglich ist, Wechselgeld für einen gegebenen Wert (z.B. 13) zurück zu geben, oder minimiere die Anzahl der Münzen
- Version 1: Unbegrenzte Anzahl von Münzen mit entsprechenden Werten
  - Rückführung auf Unbegrenztes Rucksackproblem
- Version 2: Verwende jeden Münztyp nur einmal
  - Rückführung auf 0-1-Rucksackproblem
  - Und damit auf Dynamische Programmierung

Gierige Algorithmen

- Für einige Probleme ist dynamische Programmierung des Guten zuviel
  - Gierige Algorithmen können u.U. die optimale Lösung garantieren…
  - … und sind dabei effizienter

- Beispiel: Restaurant-Platzierung
  - Sonderfall: uniformer Profit

Gierige Restaurant-Platzierung

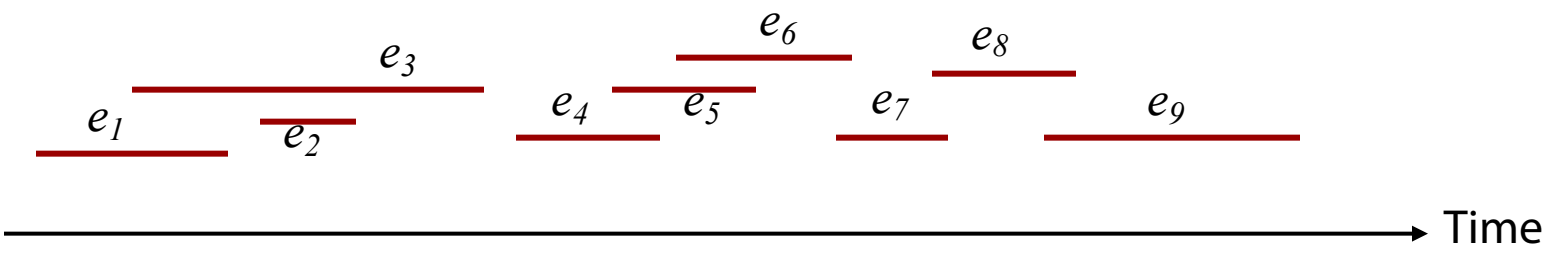



```
procedure greedy-locate([t1,…, tn], min_dist):
  select t1
  d := 0
  for i from 2 to n do
     d := d + dist(ti, ti-1)
     if d >= min_dist then
           select ti
           d := 0
  return selected towns
```

Analyse

- Zeitaufwand: $\Theta(n)$
- Speicheraufwand:
  - $\Theta(n)$ um die Eingabe zu speichern
  - $\Theta(n)$ für die gierige Auswahl

Ereignisauswahl-Problem (Uniformer Wert)

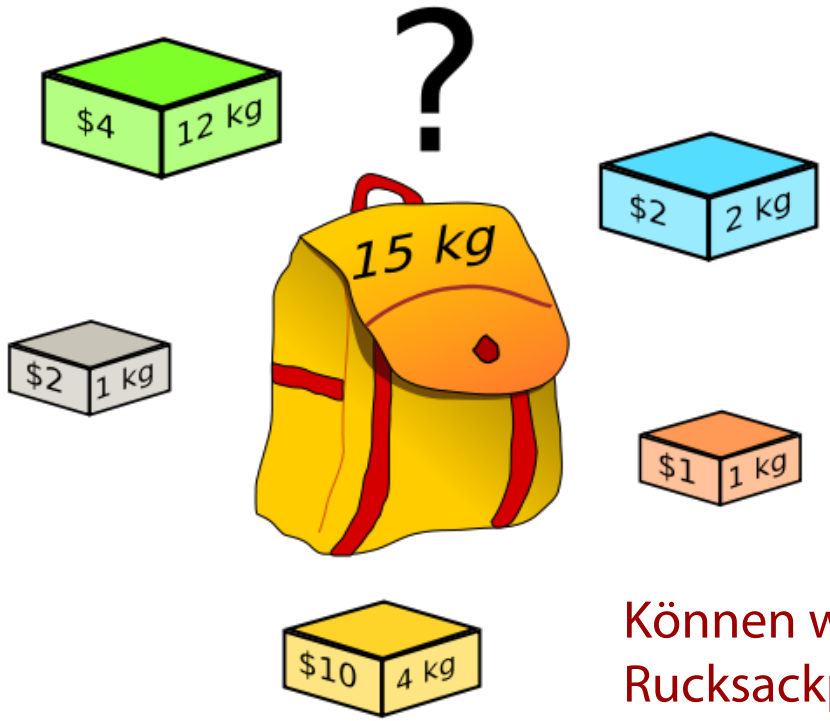


- Ziel: maximiere die **Anzahl** der ausgewählten Ereignisintervalle
- Setze $v_i = 1$ für alle i und löse Problem aufwendig mit dynamischer Programmierung
- Gierige Strategie: Wähle das nächste Ereignisintervall kompatibel mit voriger Wahl
- Liefert (e_1, e_2, e_4, e_6, e_8) für das obige Beispiel
- Warum funktioniert das hier?
  - Behauptung 1: optimale Substruktur
  - Behauptung 2: Es gibt optimale Lösung, die e_1 beinhaltet
  - Beweis durch Widerspruch: Nehme an, keine optimale Lösung enthält e_1
  - Sagen wir, das erste gewählte Ereignis ist e_i => andere gewählte Ereignisse starten, nachdem e_i endet
  - Ersetzte e_i durch e_1 ergibt eine andere optimale Lösung (e_1 endet früher als e_i)
  - Widerspruch
- Einfache Idee: Wähle das nächste Ereignis, mit dem die maximale Zeit verbleibt

Rucksackproblem

Gegeben sei eine Menge von Gegenständen

- Jeder Gegenstand hat einen Wert und ein Gewicht
- Ziel: maximiere Wert der Dinge im Rucksack
- Einschränkung: Rucksack hat Gewichtsgrenze



Drei Versionen:

0-1-Rucksackproblem: Wähle Gegenstand oder nicht

Gestückeltes Rucksackproblem: Gegenstände sind teilbar

Unbegrenztes Rucksackproblem: Beliebige Anzahlen verfügbar

Welches ist das leichteste Problem?

Können wir das gestückelte Rucksackproblem mit einem gierigen Verfahren lösen?

Gieriger Algorithmus für Stückelbares-Rucksackproblem

- Berechne Wert/Gewichts-Verhältnis für jeden Gegenstand
- Sortiere Gegenstände bzgl. ihres Wert/Gewichts-Verhältnisses
  - Verwende Gegenstand mit höchstem Verhältnis als wertvollstes Element (most valuable item, MVI)
- Iterativ:
  - Falls die Gewichtsgrenze nicht durch Addieren von MVI überschritten
    - Wähle MVI
  - Sonst wähle MVI partiell bis Gewichtsgrenze erreicht

Beispiel

Ggst	Gewicht (kg)	Wert ($)	$ / kg
1	2	2	1
2	4	3	0.75
3	3	3	1
4	5	6	1.2
5	2	4	2
6	6	9	1.5

- Gewichtsgrenze: 10

Beispiel: Sortierung

Ggst	Gewicht (kg)	Wert ($)	$ / kg
5	2	4	2
6	6	9	1.5
4	5	6	1.2
1	2	2	1
3	3	3	1
2	4	3	0.75

- Gewichtsgrenze: 10
- Wähle Ggst 5
  - 2 kg, $4
- Wähle Ggst 6
  - 8 kg, $13
- Wähle 2 kg von Ggst 4
  - 10 kg, 15.4

Wann funktioniert der gierige Ansatz?

1. Optimale Substruktur
2. Lokal optimale Entscheidung führt zur global optimalen Lösung

- Vergleiche auch die Bestimmung des minimalen Spannbaums

- Für die meisten Optimierungsprobleme gilt das nicht

Danksagung

Die nachfolgenden 5 Präsentationen wurden mit einigen Änderungen übernommen aus:

- „Algorithmen und Datenstrukturen“
  gehalten von Sven Groppe an der UzL

Gierige Strategie zur Wechselgeldberechnung

- **Beispiel:** Wechselgeldberechnung
- **Problem:** Finde eine Münzkombination für ein beliebiges Wechselgeld, die aus möglichst wenig Münzen besteht

- Quasi unbegrenztes Rucksackproblem durch Schaffnertasche

Schaffnertasche mit Münzmagazin

http://commons.wikimedia.org/wiki/File:Schaffnertasche_mit_galoppwechsler.jpeg

Gierige Strategie zur Wechselgeldberechnung

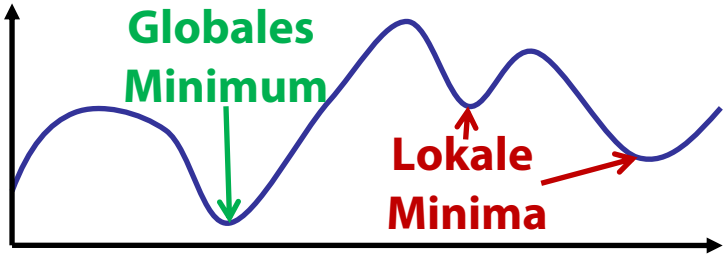
- Nimm im nächsten Schritt die größte Münze, deren Wert kleiner oder gleich dem noch verbleibenden Wechselgeld ist
- **Beispiel:**
  - Münzwerte

  - Wechselgeld sei 63 Cent:
    - 1-mal 50-Cent-Stück — Rest: 13 Cent
    - 1-mal 10-Cent-Stück — Rest: 3 Cent
    - 1-mal 2-Cent-Stück — Rest: 1 Cent
    - 1-mal 1-Cent-Stück

http://de.wikipedia.org/wiki/Eurom%C3%BCnzen

Gierige Strategie zur Wechselgeldberechnung

- Im Fall der Euro-Münzen ist die obige Strategie optimal
- Für die Münzwerte 1, 5 und 11 jedoch nicht:
  - Wechselgeld sei 15
  - Gieriges Verfahren liefert 11 1 1 1 1
  - Optimum wäre: 5 5 5
- **Damit:** Manche gierige Strategien sind anfällig dafür, ein lokales anstatt eines globalen Minimums (*bei Maximierungsproblemen: Maximums*) zu ermitteln



Gierige Strategie zur Wechselgeldberechnung

- Wie schlecht kann die Lösung der gierigen Strategie werden?
  - Münzwerte seien $n_1=1$, n_2 und n_3 mit Bedingung $n_3 = 2n_2 +1$ (vorheriges Bsp.: $n_2 = 5$, $n_3 = 11$)
  - Wechselgeld sei $N = 3n_2$ (vorheriges Bsp.: N=15)
  - **Für N sind drei Münzen optimal (3-mal n_2)**
  - **Greedy-Strategie**
    - **1-mal n_3 Rest: n_2-1**
    - **0-mal n_2 Rest: n_2-1**
    - **(n_2 -1)–mal n_1**
    - **Insgesamt n_2 Münzen**
    - **Beliebig schlecht gegenüber der optimalen Lösung (abhängig von den Münzwerten)**

Gierige Strategie zur Wechselgeldberechnung

- Es geht noch schlimmer:
  - Münzwerte seien 25 Cent, 10 Cent und 4 Cent
  - Wechselgeld sei 41 Cent
  - Lösbar ist dieses mit
    - 1-mal 25-Cent-Stück und
    - 4-mal 4-Cent-Stück
  - Greedy-Strategie
    - 1-mal 25-Cent-Stück Rest: 16 Cent
    - 1-mal 10-Cent-Stück Rest: 6 Cent
    - 1-mal 4-Cent-Stück Rest: 2 Cent
    - ??? Keine Lösung!

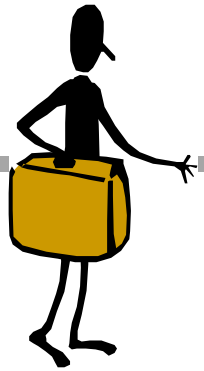
Entwurfsmuster / Entwurfsverfahren

- Im Laufe der Zeit haben sich in der immensen Vielfalt von Algorithmen nützliche Entwurfsmuster herauskristallisiert
- Gründe für das Studium von Entwurfsmustern
  - Verständnis für einzelne Algorithmen wird gesteigert, wenn man ihr jeweiliges Grundmuster verstanden hat
  - Bei der Entwicklung neuer Algorithmen kann man sich an den Grundmustern orientieren
  - Das Erkennen des zugrundeliegenden Musters hilft bei der Komplexitätsanalyse des Algorithmus
  - Warnung: Hilft **nicht** bei der Analyse der Komplexität eines **Problems**

Welches Rucksackproblem ist das leichteste?

- 0-1-Rucksackproblem: Wähle Gegenstand oder nicht
  - Schwierig
- Gestückeltes Rucksackproblem:
  Gegenstände sind teilbar
  - Leicht
- Unbegrenztes Rucksackproblem:
  Beliebige Anzahlen verfügbar
  - Es kommt auf die konkrete Probleminstanz an
  - Einige Instanzen sind leicht, andere schwierig

Zusammenfassung Entwurfsmuster



- Am Anfang des Kurses behandelt:
  - Ein-Schritt-Berechnung
  - Verkleinerungsprinzip
  - Teile und Herrsche
- Jetzt haben wir dazu noch verstanden:
  - Vollständige Suchverfahren
    - Verzweigen und Begrenzen auch genannt Branch and Bound: Dijkstra, A*,...
    - Pruning (α-β-Prinzip)
  - Suche mit richtiger Problemformulierung und Subproblemanordnung
    - Dynamisches Programmieren, Bellmans Optimalitätsprinzip (Berechnung von Teilen und deren Kombination, Wiederverwendung von Zwischenergebnissen)
  - Manchmal vollständig: Gierige Suche, Optimale Substrukturen