Rucksack packen mit TS

Vom greedy Verfahren
zur Tiefensuche

beim einfachen
Rucksackproblem

Rucksack packen mit TS

- Ausgangspunkt:

30	30	30	30
20	20	20	20

Eine Stückeliste, aus der Stücke in einen Behälter (Rucksack, Container) gegebener Größe gefüllt werden sollen.

```
(define stuecke '(30 30 30 30 20 20 20 20))
(define container '(80)
```

Rucksack packen mit TS

- Stand **`fuelle`** (greedy) Version:

```
(define
  (fuelle stuecke container)
  (cond
    ((exakt-voll? container)
     container)
    ((null? stuecke)
     #f)
    ((zu-voll? (first stuecke) container)
     (fuelle (rest stuecke) container))
    (else
     (fuelle
      (rest stuecke)
      (fuelle-ein (first stuecke) container)))
    ))
```

Mehrfachverzweigung

eine Bedingung

zugehöriger ja-Fall

else-Fall am Ende

Rucksack packen mit TS

- Hilfsfunktionen **fuellung**

```
(define (fuellung container)
  (summe (rest container))
  )
```

- und **einfuellen**

```
(define
  (fuelle-ein stueck container)
  (cons
   (first container)
   (cons stueck (rest container)))
  )
```

Rucksack packen mit TS

- Hilfsfunktion **summe**:

```
(define
  (summe inhalt)
  (cond
    ((null? inhalt)
     0)
    (else
     (+
      (first inhalt)
      (summe (rest inhalt))))
   )
)
```

(Mehrfach-) Verzweigung

Bedingung

Wert im ja-Fall

Wert im else-Fall

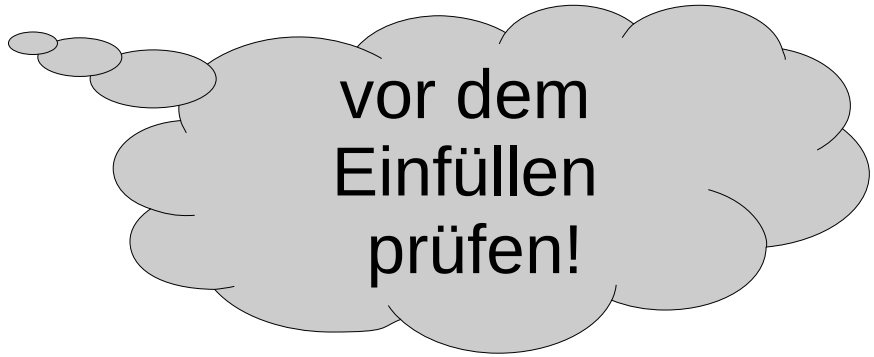
rekursiver Aufruf

Rucksack packen mit TS

- Hilfsfunktionen **`exakt-voll?`** und **`zu-voll?`**:

```
(define (exakt-voll? container)
  (= (fuellung container) (first container))
  )

(define (zu-voll? stueck container)
  (>
    (+ stueck (fuellung container))
    (first container))
  )
```



Rucksack packen mit TS

- Stand **`fuelle`** (greedy) Version:

```
(define
  (fuelle stuecke container)
  (cond
    ((exakt-voll? container)
     container)
    ((null? stuecke)
     #f)
    ((zu-voll? (first stuecke) container)
     (fuelle (rest stuecke) container))
    (else
     (fuelle
      (rest stuecke)
      (fuelle-ein (first stuecke) container)))
    ))
```

Erfolgsfall

Misserfolgsfall

Einfüllen verhindern

Rekursionsschritt

Rucksack packen mit TS

- Problem:

```
> (fuelle '(30 30 30 30 20 20 20 20) '(100))
#f
```

- Obwohl das Problem mit den gegebenen Stücken durchaus lösbar wäre. Wir wollen als Lösung bekommen:

```
> (fuelle '(30 30 30 30 20 20 20 20) '(100))
(100 30 30 20 20)
oder   (zu cons besser passend)
(100 20 20 30 30)
```

Rucksack packen mit TS

- Grund:

  Das dritte 30-er-Stück kann eingefüllt werden, verhindert aber die mögliche Füllung mit zwei 20-er-Stücken.

- Lösungsidee:

  Wir müssen zu einem vorigen Zustand zurück gehen können und von dort aus die Alternativen prüfen.

  → Idee des *backtracking*

Rucksack packen mit TS

- Lösungsansatz:

  Der Rekursionsschritt in die Tiefe darf nur bedingt erfolgen.

- „*Normale*“ Realisierung wäre eine interne Verzweigung im bisherigen else-Fall.
- Scheme lässt im **cond** aber zu, dass eine Bedingung gleichzeitig den Wert definiert. Dann darf sie kein Paar sein, sondern sie muss allein in der Klammer stehen.

Rucksack packen mit TS

- Erweiterung **`fuelle`** Tiefensuche-Version:

```
(define
  (fuelle stuecke container)
  (cond
    ((exakt-voll? container)
     container)
    ((null? stuecke)
     #f)
    ((zu-voll? (first stuecke) container)
     (fuelle (rest stuecke) container))
    (else
     (fuelle
      (rest stuecke)
      (fuelle-ein (first stuecke) container)))
    (else …)
 ))
```

Erfolgsfall

Misserfolgsfall

Einfüllen verhindern

Versuch des Rekursionsschritts

Rucksack packen mit TS

- Erweiterung **`fuelle`** Tiefensuche-Version:

```
(define
  (fuelle stuecke container zielgroesse)
  (cond
    . . .
    ((zu-voll? container zielgroesse)
     (fuelle stuecke (rest container) zielgroesse))
    ((fuelle
      (rest stuecke)
      (fuelle-ein (first stuecke) container
      zielgroesse))
    (else
      (fuelle
       (rest stuecke)
       container
       zielgroesse))
 ))
```

Einfüllen verhindern

Versuch des Rekursionsschritts mit dem Stück

Alternativer Rekursionsschritt ohne das Stück

Rucksack packen mit TS

Ablauf:

- nicht exakt voll, noch Stücke da, nicht zu voll
  → erstes 30-er-Stück wird versucht
- nicht exakt voll, noch Stücke da, nicht zu voll
  → zweites 30-er-Stück wird versucht
- nicht exakt voll, noch Stücke da, nicht zu voll
  → drittes 30-er-Stück wird versucht
- nicht exakt voll, noch Stücke da, aber zu voll
  → viertes 30-er-Stück wird verworfen
- nicht exakt voll, noch Stücke da, aber zu voll
  → erstes 20-er-Stück wird verworfen

bis

- nicht exakt voll, keine Stücke mehr da
  → #f zurück geben über vier Stufen und mit weiteren 20-er Versuchen
- else ohne das dritte 30-er-Stück versuchen
- nicht exakt voll, noch Stücke da, nicht zu voll
  → erstes 20-er-Stück wird versucht
- nicht exakt voll, noch Stücke da, nicht zu voll
  → zweites 20-er-Stück wird versucht
- exakt voll
  → (100 20 20 30 30) zurück geben

Rucksack packen mit TS

- *Hinweis: Reihenfolge beim Schritt in Tiefe egal*

```
(define
  (fuelle stuecke container zielgroesse)
  (cond
    . . .
    ((zu-voll? container zielgroesse)
     (fuelle stuecke (rest container) zielgroesse))
    ((fuelle
      (rest stuecke)
      container
      zielgroesse))
    (else
      (fuelle
       (rest stuecke)
       (fuelle-ein (first stuecke) container)
       zielgroesse))
 ))
```

Einfüllen verhindern

Versuch des Rekursionsschritts ohne das Stück

Alternativer Rekursionsschritt mit dem Stück