Entwicklung der Tiefensuche am Rucksackproblem

```
;;;                                                          "rucksack-ts-entwicklung-2010.scm"
;;; Demonstration der Entwicklung des Programmes zur Tiefensuche am Rucksackproblem
(load "../breakpoint.scm")
; ----- Liste der vorhandenen Stuecke -----
(define stuecke '(30 30 30 20 20 20))
```

Hilfsfunktionen

```
;;; ===== ANFANG HILFSFUNKTIONEN =====
;;; ----- Hilfsfunktion fuellung -----;
; (ersetzbar durch Anwendung von apply + )
(define
  (fuellung container)
  (if
     (null? container)
     0
     (+ (first container) (fuellung (rest container)))))
;;; verwendet von:
; ----- Praedikate -----
;;; ----- Hilfsfunktion exakt-voll? -----
; prüft, ob der Container exakt voll ist.
(define
  (exakt-voll? container zielgroesse)
  (= (fuellung container) zielgroesse)
  )
;;; alternativ ohne die Funktion fuellung:
(define
  (exakt-voll? container zielgroesse)
  (= (apply + container) zielgroesse)
  )
```

```
;;; ----- Hilfsfunktion zu-voll? -----
; prüft, ob der container zu voll ist.
(define
  (zu-voll? container zielgroesse)
  (> (fuellung container) zielgroesse)
  )

(define
  (zu-voll? container zielgroesse)
  (> (apply + container) zielgroesse)
  )
```

Funktionskopf

```
; 1. Schritt: Bestimmung des Funktionskopfes
(define
  (tiefensuche stuecke container zielgroesse)
  'wert-undefiniert)



; Testaufrufe:

(ausgabe "\nSchritt 1: ")                    Schritt 1:
(tiefensuche stuecke '() 80)                 wert-undefiniert
(tiefensuche stuecke '() 100)                wert-undefiniert
```

Erste Verzweigung einarbeiten

```
;;; ----- exakt gefüllt? -----
; 2. Schritt: ich bin am Ziel, der Container ist exakt gefuellt.
; Hier ist also eine Bedingung zu überprüfen. --> cond
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; Bedingung
     container)                                ; Erfolgs-Fall: container zurueckgeben
    (else
     'wert-undefiniert)                        ; sonst-Fall
   )
  )



; Testaufrufe:
(ausgabe "\nSchritt 2: ")                              Schritt 2:
(tiefensuche stuecke '(20 20 20 20) 80)                (20 20 20 20)
(tiefensuche stuecke '() 80)                           wert-undefiniert
(tiefensuche stuecke '() 100)                          wert-undefiniert
```

Bei Rekursion mit Abbau einer Liste immer auf leere Liste testen

Grundsätzlich sollte man zwar diesen Test immer zuerst durchführen. Sollte aber zufällig gerade das letzte Stück den Rucksack [Container] exakt füllen, wollen wir diesen Erfolgsfall natürlich nicht verlieren.

```
;;; ----- am "Ende", aber nicht am Ziel -----
; 3. Schritt: es geht nicht weiter, keine Stuecke mehr!
; Alle Stuecke wurden versucht, ohne dass sich der Container passend füllen ließ.
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; Bedingung
     container)                                ; Erfolgs-Fall: container zurueckgeben
    ((null? stuecke)                           ; 2.Bedingung
     #f)                                       ; Misserfolgs-Fall. #f kennzeichnet Misserfolg.
    (else                                      ; und sonst?
     'wert-undefiniert)                        ; der "sonst"-Fall
    )
  )
```

```
; Testaufrufe:
(ausgabe "\nSchritt 3: ")                      Schritt 3:
(tiefensuche stuecke '(20 20 20 20) 80)        (20 20 20 20)
(tiefensuche '() '(30 30) 80)                  #f
(tiefensuche stuecke '() 80)                   wert-undefiniert
(tiefensuche stuecke '() 100)                  wert-undefiniert
```

Wenn der Rucksack zu voll wird, ist das auch ein Misserfolg

```
;;; ----- zu voll -----
; 4. Schritt: der Container ist zu voll.
; Misserfolg signalisieren
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; Bedingung
     container)                                ; Erfolgs-Fall: container zurueckgeben
    ((null? stuecke)                           ; 2.Bedingung
     #f)                                       ; Misserfolgs-Fall. #f kennzeichnet Misserfolg.
    ((zu-voll? container zielgroesse)          ; prüfen
     #f)                                       ; so geht es nicht. #f kennzeichnet Misserfolg.
    (else                                      ; und sonst?
     'wert-undefiniert)                        ; der "sonst"-Fall
    )
  )
```

```
; Testaufrufe:
(ausgabe "\nSchritt 4: ")                          Schritt 4:
(tiefensuche stuecke '(20 20 20 20) 80)            (20 20 20 20)
(tiefensuche '() '(30 30) 80)                      #f
(tiefensuche '(20 20 20) '(30 30 30) 80)           #f
(tiefensuche stuecke '() 80)                       wert-undefiniert
(tiefensuche stuecke '() 100)                      wert-undefiniert
```

Wenn wir alle Möglichkeiten haben, kommt nun der else - Fall

```
;;; ----- der sonst-Fall ??? -----
; 5. Schritt: Es wird weitergesucht mit dem aktuellen Stueck im Container.
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; Bedingung
     container)                                ; Erfolgs-Fall: container zurueckgeben
    ((null? stuecke)                           ; 2.Bedingung
     #f)                                       ; Misserfolgs-Fall. #f kennzeichnet Misserfolg.
    ((zu-voll? container zielgroesse)          ; prüfen
     #f)                                       ; so geht es nicht. #f kennzeichnet Misserfolg.
    (else                                      ; und sonst?
     (tiefensuche                              ; weiter suchen:.
      (rest stuecke)                           ; das erste Element wechselt ...
      (cons (first stuecke) container)         ; ... in den Container;
      zielgroesse))                            ; es bleibt beim selben Zielwert.
    )
  )



; Testaufrufe:
(ausgabe "\nSchritt 5: ")                              Schritt 5:
(tiefensuche stuecke '(20 20 20 20) 80)                (20 20 20 20)
(tiefensuche '() '(30 30) 80)                          #f
(tiefensuche '(20 20 20) '(30 30 30) 80)               #f
(tiefensuche stuecke '() 80)                           #f
(tiefensuche stuecke '() 100)                          #f
```

Wir hatten aber noch nicht alle Möglichkeiten!

```
;;; ----- das war es noch nicht ! -----
; 6. Schritt: das Stueck einzupacken fuehrte ggf. in eine Sackgasse.
; Wir muessen es auch ohne dies Stueck versuchen.
; Dazu muss der Aufruf mit dem Stueck (das ist der Schritt in die Tiefe)
; wie eine Bedingung behandelt werden. Wie das z.B. geht, s.u.
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; Bedingung
     container)                                ; Erfolgs-Fall: container zurueckgeben
    ((null? stuecke)                           ; 2.Bedingung
     #f)                                       ; Misserfolgs-Fall. #f kennzeichnet Misserfolg.
    ((zu-voll? container zielgroesse)          ; prüfen
     #f)                                       ; so geht es nicht. #f kennzeichnet Misserfolg.
     ((tiefensuche                             ; ersuch des Schrittes in die Tiefe ...
      (rest stuecke)                           ; das erste Element wechselt ...
      (cons (first stuecke) container)         ; ... in den Container;
      zielgroesse))                            ; es bleibt beim selben Zielwert.
    ;;; Eine der besonderen Moeglichkeiten von cond: Bedingung=Wert
    ;;; Die Bedingung hat einen Wert, der zurückgegeben werden kann.
    ;;; Bei erfolglosem Versuch brauchen wir dann hier den Wert #f,
    ;;; damit zum else gegangen wird.
    (else                                      ; und sonst?
     (tiefensuche                              ; Wir müssen weiter suchen ...
      (rest stuecke)                           ; ... ohne das erste Element ...
      container                                ; ... mit der bisherigen Fuellung
      zielgroesse))                            ; ... und dem selben Zielwert.
    )
  )
```

Das vollständige Programm mit kommentierter Ausgabe und Testaufrufe dazu

```
;;; ----- Vervollständigung der Ausgabe -----
; 7. Schritt: kommentierte Ausgabe!
(define
  (tiefensuche stuecke container zielgroesse)
  (cond
    ((exakt-voll? container zielgroesse)       ; 1. Bedingung
     (ausgabe "Fuellung " zielgroesse " erreicht mit Container " container)
     container)                                ; Erfolgs-Fall
    ((null? stuecke)                           ; 2.Bedingung
     #f)                                       ; Misserfolgs-Fall. #f kennzeichnet Misserfolg.
    ((zu-voll? container zielgroesse)          ; prüfen
     #f)                                       ; so geht es nicht. #f kennzeichnet Misserfolg.
     ((tiefensuche                             ; ersuch des Schrittes in die Tiefe ...
      (rest stuecke)                           ; das erste Element wechselt ...
      (cons (first stuecke) container)         ; ... in den Container;
      zielgroesse))                            ; es bleibt beim selben Zielwert.
    (else                                      ; und sonst?
     (tiefensuche                              ; Wir müssen weiter suchen ...
      (rest stuecke)                           ; ... ohne das erste Element ...
      container                                ; ... mit der bisherigen Fuellung
      zielgroesse))                            ; ... und dem selben Zielwert.
    )
  )

; Testaufrufe:
(ausgabe "\nSchritt 7: ")  Schritt 5:
(tiefensuche stuecke '(20 20 20 20) 80)              Fuellung  80  erreicht mit Container  (20 20 20 20)
(tiefensuche '() '(30 30) 80)                        #f
(tiefensuche '(20 20 20) '(30 30 30) 80)             #f
(tiefensuche stuecke '() 80)                         Fuellung  80  erreicht mit Container  (20 30 30)
(tiefensuche stuecke '() 100)                        Fuellung  100  erreicht mit Container  (20 20 30 30)
```