Tiefensuche arbeitet mit einem Stack

Die Darstellung des Aufrufstacks im Debugger-Fenster erleichtert das Verständnis des Vorgehens bei der Tiefensuche und der Datenstruktur Stack. Beispielhaft wird hier eine Programm zum Füllen von zwei Containern (Rucksack-Problem) mit gleicher Zielgröße betrachtet.

```
; fuelle-zwei-ts.scm
;;; ===== ANFANG HILFSFUNKTIONEN =====
(define
; prüft, ob das Stück exakt in den Container passt.
  (exakt-voll? container zielgroesse)
  (= (apply + container) zielgroesse)
  )
(define
; prüft, ob der Container (Rucksack) die Zielgroesse überschreitet.
  (zu-voll? container zielgroesse)
  (> (apply + container) zielgroesse))
;;; ===== ENDE HILFSFUNKTIONEN =====

(define
  (tiefensuche stuecke container-1 container-2 zielgroesse)
  (cond
    ((and                                    ; Ziel beide voll erreicht
      (exakt-voll? container-1 zielgroesse)
      (exakt-voll? container-2 zielgroesse))
     (list  container-1 container-2))        ; Ergebnis ausgeben

    ((null? stuecke)                         ; Misserfolgs-Fall
     #f)                                     ; #f kennzeichnet Misserfolg.

    ((zu-voll? container-1 zielgroesse)      ; so geht es nicht bei container-1
     #f)                                     ; #f kennzeichnet Misserfolg.

    ((zu-voll? container-2 zielgroesse)      ; so geht es nicht bei container-2
     #f)                                     ; #f kennzeichnet Misserfolg.

    ((tiefensuche                            ; 1.Versuch Schritt in die Tiefe
      (rest stuecke)
      (cons (first stuecke) container-1)     ; ... mit dem stueck in container-1
      container-2
      zielgroesse))                          ; wenn nicht erfolgreich:

    ((tiefensuche                            ; 2.Versuch Schritt in die Tiefe
      (rest stuecke)
      container-1
      (cons (first stuecke) container-2)     ; ... mit dem stueck in container-2
      zielgroesse))
    (else                                    ; wenn auch das nicht erfolgreich:

     (tiefensuche                            ; müssen wir weiter suchen ...
      (rest stuecke)                         ; ... ohne das aktuelle Stueck ...
      container-1                            ; ... mit den bisherigen Fuellungen
      container-2
      zielgroesse))                          ; ... und derselben Zielgroesse.
    )
  )
```

Mit dem Testaufruf

```
(tiefensuche '(60 40 30 30 30 20 20 20) '() '() 80)
```

erhalten wir die Lösung

```
((20 60) (20 20 40))
```

Der Stack

Bei jedem Schritt in die Tiefe muss sich das Programm den aktuellen Zustand merken. Dazu gehören die aktuellen Werte der Variablen und die Programmposition. Sie müssen auf einem Stack abgelegt werden, damit nach dem Rücksprung die Werte der Variablen mit den Werten von Stack wiederhergestellt werden können.
Der Stack, deutsch Stapel, ist eine Datenstruktur bei der die Zugriffe immer *"oben am Kopf*" erfolgen, also oben auf abgelegt und auch von oben wieder entfernt. Man bezeichnet das Zugriffsprinzip als ***Last-In-First-Out*** oder abgekürzt ***lifo***.

Die Arbeit auf dem Stack erledigt das System

Beim Ausführen des o.a. Aufrufs merkt sich das System in seinem Aufrufstack, dass dieser Aufruf bearbeitet werden soll und den aktuellen Wert der Parameter:
'(60 40 30 30 30 20 20 20) '() '() 80
Im cond-Bereich werden zunächst die Abbruchbedingungen bearbeitet und dann der erste Schritt in die Tiefe ausgeführt, bei dem sich das System dessen Programmposition merken muss, um später dort weiter arbeiten zu können und die Werte der Variablen.
Auf der nächsten Rekursionsebene sind diese nun durch
'(40 30 30 30 20 20 20) '(60) '() 80
belegt.
Wieder werden die Abbruchbedingungen bearbeitet und dann wieder das erste Stück in den ersten Container eingefüllt, Zustand merken und Aufruf mit
'(30 30 30 20 20 20) '(40 60) '() 80
Der Zustand führt bei der Abbruchbedingung **zu-voll?** für den ersten Container zu einem Rücksprung in die vorherige Rekursionsebene, wobei durch Aufruf vom Stack der Wert
'(40 30 30 30 20 20 20) '(60) '() 80
wiederhergestellt und der nachfolgende Aufruf mit dem aktuell ersten Stück im zweiten Container versucht wird.
'(30 30 30 20 20 20) '(60) '(40) 80
ist zulässig, daher wird wieder in die Tiefe gegangen mit:
'(30 30 20 20 20) '(30 60) '(40) 80
Unzulässig → Rücksprung zu
'(30 30 30 20 20 20) '(60) '(40) 80
und im zweiten Container versuchen
'(30 30 20 20 20) '(60) '(30 40) 80
Zulässig, Schritt in die Tiefe mit aktuellen Stück im ersten Container:
'(30 20 20 20) '(30 60) '(30 40) 80
Unzulässig → Rücksprung zu
'(30 30 20 20 20) '(60) '(30 40) 80
Schritt in die Tiefe mit aktuellen Stück im zweiten Container:
'(30 20 20 20) '(60) '(30 30 40) 80
Unzulässig, Rücksprung zum Zustand
'(30 30 20 20 20) '(60) '(30 40) 80
und else bearbeiten, also Schritt in die Tiefe ohne das Stück:
'(30 20 20 20) '(60) '(30 40) 80
Nach einigen weiteren Schritten wird auch das weiter 30-Stück verworfen.
'(20 20 20) '(60) '(30 40) 80

Der Schritt in die Tiefe ergibt nun im ersten Container einen Erfolgsfall, allerdings nicht im zweiten. Das gilt wegen der Stücke mit gleichem Wert für alle nachfolgenden Schritte in die Tiefe, so dass dann die Abbruchbedingung **`(null? stuecke)`** erfüllt ist.
Erst nach sehr vielen weiteren Versuchen wird beim Rücksprung zur zweiten Ebene das Einfüllen aller **`30`**-Stücke verworfen und die Lösung **`((20 60) (20 20 40))`** erzielt.
Bei den hier vorliegenden Werten wird diese Lösung in einem Blatt des Rekursionsbaums erzielt, also beim Zustand der leeren Stückeliste.
Das ist beim vorliegenden Programm allerdings nicht zwingend, da das Programm mit einer erfolgreichen Lösung alle Rekursionsebenen verlässt und den gewonnenen Wert als Wert des ersten Aufrufs zurückgibt.
Grundsätzlich kann man nach dem Prinzip der vollständige Suche den gesamten Rekursionsbaum jeweils immer bis zur größten Tiefe durchlaufen und erst dann prüfen, ob die Lösung zulässig ist. Das hier vorliegende Programm ist also bereits eine optimierte vollständige Suche.
Hinweis: Sollen alle unzulässigen und zulässigen Lösungen bestimmt werden, muss der Baum vollständig durchlaufen werden.

Aufgabe:

Berechnen Sie die Anzahl der Blätter bei einer vollständigen Suche für den gegebenen Aufruf.