Tiefensuche arbeitet mit einem Stack

Die Darstellung des Aufrufstacks erleichtert das Verständnis des Vorgehens bei der Tiefensuche und der Datenstruktur Stack. Beispielhaft wird hier ein Programm zum Füllen von einem Container (Rucksack-Problem) betrachtet.

```
# Globale Variable für die Stücke:
stuecke=[30,30,30,20,20,20]

# Prädikate
def istVoll(container, fassungsvermoegen):
    return fassungsvermoegen==sum(container)

def wirdZuVoll(stueck, container, fassungsvermoegen):
    return fassungsvermoegen < stueck+sum(container)

# die eigentliche Suchfunktion
def fuelle(stuecke, container, fassungsvermoegen):
    if istVoll(container, fassungsvermoegen):
        return container
    if len(stuecke)==0:
        return False
    if wirdZuVoll(stuecke[0], container, fassungsvermoegen):
        return False
    ergebnis=fuelle(stuecke[1:],container+stuecke[:1], fassungsvermoegen)
    if ergebnis:
        return ergebnis
    else:
        return fuelle(stuecke[1:], container, fassungsvermoegen)
```

Mit dem Testaufruf

```
fuelle(stuecke, [], 100)
```

erhalten wir die Lösung

```
[30, 30, 20, 20]
```

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 abgelegt wird 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 zu den Parametern jeweils ihren aktuellen Wert:

```
[30,30,30,20,20,20], [], 100
```

Im Verzweigungs-Bereich werden zunächst die Abbruchbedingungen bearbeitet und dann der erste Schritt in die Tiefe ausgeführt, bei dem sich das System dessen Programm-

position merken muss, um später dort weiter arbeiten zu können und die aktuellen Werte der Parameter.

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**

Auf der nächsten Rekursionsebene sind diese nun durch
[30,30,20,20,20], [30], 100
belegt,
Wieder werden die Abbruchbedingungen bearbeitet und dann wieder das erste Stück in den Container eingefüllt, Zustand merken

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**

und Aufruf mit
[30,20,20,20], [30,30], 100

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**
2	**[30,20,20,20]**	**[30,30]**	**100**

Auf der folgenden Rekursionsebene sind diese nun durch
[20,20,20], [30,30,30], 100
belegt.

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**
2	**[30,20,20,20]**	**[30,30]**	**100**
3	**[20,20,20]**	**[30,30,30]**	**100**

Dieser Zustand führt bei der Abbruchbedingung für den zu füllenden Container
if wirdZuVoll(stuecke[0], container, fassungsvermoegen) zu einem Rücksprung mit **False** in die vorherige Rekursionsebene, wobei durch Aufruf vom Stack die Werte (s.o.)
[30,20,20,20], [30,30], 100
wiederhergestellt werden und der nachfolgende Aufruf (*else-Fall*) ohne das aktuell erste

Stück im Container also mit den Werten **`[20,20,20], [30,30], 100`** versucht wird.
Der Stack nimmt also den folgenden Zustand ein:

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**
2	**[30,20,20,20]**	**[30,30]**	**100**
3	**[20,20,20]**	**[30,30]**	**100**

und das Einfügen ist zulässig, daher wird wieder in die Tiefe gegangen mit:
`[20,20], [30,30,20], 100`

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**
2	**[30,20,20,20]**	**[30,30]**	**100**
3	**[20,20,20]**	**[30,30]**	**100**
4	**[20,20]**	**[30,30,20]**	**100**

Der nächste Schritt in die Tiefe mit **`[20], [30,30,20,20], 100`**
ergibt nun für den Container den Erfolgsfall.

Stufe	stuecke	Container	fas...
0	**[30,30,30,20,20,20]**	**[]**	**100**
1	**[30,30,20,20,20]**	**[30]**	**100**
2	**[30,20,20,20]**	**[30,30]**	**100**
3	**[20,20,20]**	**[30,30]**	**100**
4	**[20,20]**	**[30,30,20]**	**100**
5	**[20]**	**[30,30,20,20]**	**100**

Der Stack kann nun rückwärts abgebaut werden, wobei jeweils beim Rücksprung keine weitere Bearbeitung erfolgt und die Rückgabe ist die gewünschte Belegung.
Diese Art der Bearbeitung wird als Endrekursion bezeichnet, die leider von Python nicht erkannt wird. Wäre das anders, könnte auf den Einsatz des Stacks verzichtet werden und statt dessen würde der jeweilige Aufruf den vorherigen vollständig ersetzen.

Aufgabe:

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