# Tiefensuche und Breitensuche

**Tiefensuche, ein Fall von bruteforce**

Eines der Verfahren, die mit „brute force“ eine Lösung zu finden versuchen, ist die Tiefensuche. Bei der Tiefensuche wird immer der erste noch nicht untersuchte Ast des Baumes[1] weiter verfolgt. Wird er verworfen, dann geht man im Entwicklungsbaum einen Schritt zurück und schaut sich dort nach Alternativen um. Findet man eine Alternative, dann versucht man diese und verfolgt den Baum auch hier nach dem selben Prinzip.

Hier ist zu erkennen: Eine typisch rekursive Beschreibung des Problems drängt sich auf!

### *Tiefensuche mit backtracking*

Typisch für rekursive Beschreibung von Problemen ist, dass man nicht einen Ablauf beschreibt, sondern das, was man auf einer Rekursionsstufe zu tun hat und die verschiedenen Möglichkeiten, auf die man auf dieser Rekursionsstufe treffen kann.

Aufgabe: Untersuchen Sie:

*Was ist bei einer rekursiven Tiefensuche beim Eintreten in die Rekursionsstufe zu tun und welche Fälle sind zu unterscheiden?*

Beim Eintreten in die Rekursionsstufe sind alle Alternativen zu bestimmen, die es hier gibt. Man spricht von einer Expansion (Graphen: ein Knoten wird expandiert). Dafür muss uns eine Funktion zur Verfügung stehen, die uns die möglichen Alternativen bestimmt. Typisch für die Tiefensuche ist, dass von diesen mit Hilfe einer Nachfolgerfunktion bestimmten Alternativen zunächst nur eine weiter verfolgt wird, ehe man sich mit den anderen beschäftigt.

Bei einer Suche haben wir – genau so wie bei jeder Rekursion – zunächst die Fälle zu untersuchen, die zu einem Rekursionsabbruch führen:

- Es gibt keine Alternativen (mehr). Dies bedeutet für die Suche den Misserfolgsfall und es ist das backtracking auszulösen und ein Wert zurückzugeben, der auf der aufrufenden Rekursionsstufe als Misserfolgsfall interpretiert werden kann. Das kann z.B. der boolean – Wert `False` sein oder eine leere Liste o.ä.
- Die erste Alternative stellt den Zielwert dar. Damit hat unsere Suche zu einem Erfolg geführt. Es muss auch in diesem Fall ein Rekursionsabbruch erfolgen, allerdings mit einem Rückgabewert, der den Erfolgsfall kennzeichnet. In vielen Fällen wird man außerdem als Nebeneffekt eine Ausgabe z.B. auf dem Bildschirm haben wollen, bei welcher der erfolgreiche Suchpfad ausgegeben wird.
- Die erste Alternative kommt schon im Suchpfad vor. Wir haben bei unserer Suche einen Zyklus gefunden. Bei Suchbäumen tritt dieser Fall natürlich nicht auf, in vielen Suchräumen muss man aber mit Zyklen rechnen. Die Tiefensuche muss mit dem Problem fertig werden, sie versackt sonst in einer Endlosschleife. Das Mittel dazu ist das Führen einer Besuchtliste. Auch dies bedeutet für die Suche den Misserfolgsfall und es ist das backtracking auszulösen und ein Wert zurückzugeben, der auf der aufrufenden Rekursionsstufe als Misserfolgsfall interpretiert werden kann.
- Führe den Rekursionsaufruf für die erste Alternative durch. Dieser Aufruf ist ein normal rekursiver Aufruf. Zu beachten ist, dass er zu einem Wert führt, der in irgendeiner

1 In unserem Fall ist es ein Baum. Das ist leider jedoch nicht zwingend, wie wir noch sehen werden.

angemessenen Weise interpretiert werden muss. Bekommt man von der aufgerufenen Stufe Erfolg gemeldet, ist ggf. als Nebeneffekt eine Ausgabe o.ä. zu machen. Abhängig von der Frage, ob man eine vollständige Suche durchführt oder nicht, sind danach noch die Alternativen zu behandeln. Handelt es sich um eine einfache Suche, ist auch auf dieser Stufe der Wert zurückzugeben, der den Erfolgsfall kennzeichnet.

- Führe den Aufruf der restlichen Alternativen durch. Dieser Rekursionsaufruf erfolgt nun allerdings endrekursiv, wir dürfen also in der Rekursionsebene nicht weiter absteigen.

Aufgabe:

*Schreiben Sie das Python – Programm!*

### *Lösungen und Probleme*

Bei diesem Problem ist der Suchraum ein Baum, da es auf jeder Stufe immer nur die Alternativen *"das Stück einfüllen"* oder *"das Stück nicht einfüllen"* gibt.
Eine einfache Lösung, aber schon optimierte Lösung könnte so aussehen[1]:

```
def fuelle(stuecke, container, fassungsvermoegen):
    if fassungsvermoegen==sum(container):
        return container
    elif len(stuecke)==0:
        return False
    elif fassungsvermoegen<sum(container):
        return False
    else:
        ergebnis=fuelle(stuecke[1:], container+stuecke[:1], fassungsvermoegen)
        if ergebnis:
            return ergebnis
        else:
            return fuelle(stuecke[1:], container, fassungsvermoegen)
```

Alternativen sind tatsächlich nur die Möglichkeiten hinein oder nicht hinein. Wir haben hier eben einen binären Suchgraphen vorliegen, daher geht es an zweiter Stelle nicht um das Abbauen mehrerer Alternativen auf der selben Stufe.

#### Typisch

Typisch für diese Aufrufstruktur ist, dass vor dem Schritt in die Tiefe die Alternativen –hier ist es eine einzige– nicht wirklich bestimmt werden, sondern ihr Erzeugen nur angelegt wird. Python legt den noch möglichen Aufruf –möglich deswegen, weil er nur erfolgt, wenn der Schritt in die Tiefe nicht erfolgreich war– auf dem Systemstack ab und führt ihn ggf. erst beim Rücksprung, also beim backtracking aus.

#### Systemstack

Im Systemstack verwaltet Python ohne unser Zutun den Programmzustand auf jeder Rekursionsstufe, also welchen Wert die Parameter aktuell haben und wo wir uns im Funktionsablauf befinden.

Das eigentliche Problem von Python ist aber, dass nicht erkannt wird, wenn Aufrufe endrekursiv sind und so auch dann "Müll" auf dem stack hinterlassen wird, wenn der endrekursive Aufruf eigentlich den vorigen ersetzen könnte. Python arbeitet also in jedem Fall rekursiv nachklappernd.

1 Die Reihenfolge stimmt nicht mit der o.a. überein.

***Tiefensuche ist bruteforce***

Tiefensuche ist ein vollständiges Suchverfahren. Es sucht alle Möglichkeiten ab und geht dabei in dem Fall der nicht optimierten Tiefensuche in diesem Beispiel bei z.B. 20 Stücken insgesamt $2^{20} \approx 10^6$ den gesamten Suchbaum durch. Unser hier vorliegendes Programm optimiert diese Suche dadurch, dass die Rekursion

- einerseits beim ersten Erfolgsfall und
- andererseits beim Überschreiten der zulässigen Füllung

vorzeitig abgebrochen wird und nicht mehr sinnvolle Alternativen auch nicht mehr untersucht werden.

***Breitensuche, ein weiterer Fall von bruteforce***

Den Unterschied zwischen Tiefensuche und Breitensuche zu beschreiben und zu verstehen ist sehr einfach. Die Stichworte sind „Tiefe zuerst“ oder „Breite zuerst“. Bei der Breitensuche muss erreicht werden, dass erst alle Alternativen einer Stufe bearbeitet werden, bevor eine tiefere Stufe angegangen wird. Bei diesem kleinen Unterschied ist die erste Vermutung, dass man mit demselben Programm bei geringfügigen Änderungen zur Lösung kommen müsste. Das ist jedoch leider nicht der Fall. Während uns bei der Tiefensuche der Systemstack die Sicherung der möglichen Alternativen bis zum Wiedereintreten in die aktuelle Ebene übernimmt – Daten und Programmposition werden wieder in den vorigen Zustand versetzt – müssen wir uns bei der Breitensuche selbst darum kümmern.

Oben heißt es: Bei der Breitensuche muss erreicht werden, dass erst alle Alternativen einer Stufe bearbeitet werden, bevor eine tiefere Stufe angegangen wird.

Die dazu verwendete Datenstruktur heißt Warteschlange (queue). Der Unterschied zwischen **stack** und **queue** wird schlagwortartig formuliert durch LIFO und FIFO:

Bei der Tiefensuche heißt es:

*Zum Speichern der Möglichkeiten muss eine **LIFO** - Struktur (= **l**ast **i**n **f**irst **o**ut), also eine **stack** - Struktur (Stapel) benutzt werden.*

Bei der Breitensuche muss es nun heißen:

*Zum Speichern der Möglichkeiten muss eine **FIFO** - Struktur (= **f**irst **i**n **f**irst **o**ut), also eine **queue** - Struktur (Warteschlange) benutzt werden.*

Geht man davon aus, dass z.B. immer vorn geleert wird, dann unterscheiden sich beide in der Position, an der gefüllt wird: Beim stack wird immer vorn eingebaut *und* abgebaut, bei der Warteschlange wird zwar auch *vorn* abgebaut, aber immer *hinten* eingebaut!

Sonst ist das Programm tatsächlich vergleichbar. Zu der zu bearbeitenden Alternative werden ihre Nachfolger bestimmt und bearbeitet oder gespeichert.

Aufgabe zum Vergleich von Tiefensuche und Breitensuche

- Welche Vorteile und Nachteile bietet die Tiefensuche?
- Formulieren und Probieren!

Spätestens hier sollte man einmal auf eine etwas schwierigere Variante des Rucksack-Problems oder eine einfache CLP – Variante wechseln.