8.3 Aufgaben

8.3.1 Aufgabe 1: Türme von Hanoi (★★★☆☆)

Beim Türme-von-Hanoi-Problem gibt es drei Türme oder Stäbe A, B und C. Zu Beginn sind mehrere gelochte Scheiben der Größe nach auf Stab A platziert, die größte zuunterst. Ziel ist es nun, den gesamten Stapel, also alle Scheiben, von A nach C zu bewegen. Dabei darf immer nur eine Scheibe nach der anderen bewegt werden und niemals eine kleinere Scheibe unter einer größeren liegen. Deswegen benötigt man den Hilfsstab B. Schreiben Sie eine Methode `void solveTowersOfHanoi(int)`, die die Lösung auf der Konsole in Form der auszuführenden Bewegungen ausgibt.

Beispiel Das Ganze sieht in etwa wie folgt aus:

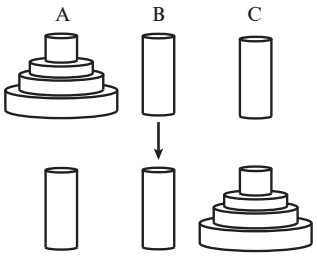

Abbildung 8-1 *Aufgabenstellung beim Türme-von-Hanoi-Problem*

Für drei Scheiben soll der folgende Lösungsweg ausgegeben werden:

```
Tower Of Hanoi 3
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

Bonus Erstellen Sie eine konsolenbasierte grafische Aufbereitung. Für zwei Scheiben würde dies etwa wie folgt aussehen:

```
Tower Of Hanoi 2
   A         B         C
   |         |         |
  #|#        |         |
 ##|##       |         |
-------   -------   -------
Moving slice 1: Tower [A] -> Tower [B]
   A         B         C
   |         |         |
   |         |         |
 ##|##      #|#        |
-------   -------   -------
```

```
Moving slice 2: Tower [A] -> Tower [C]
   A         B         C
   |         |         |
   |         |         |
   |        #|#      ##|##
-------   -------   -------
Moving slice 1: Tower [B] -> Tower [C]
   A         B         C
   |         |         |
   |         |        #|#
   |         |       ##|##
-------   -------   -------
```

8.3.2 Aufgabe 2: Edit Distance (★★★★☆)

Berechnen Sie für zwei Strings, wie viele Änderungen diese – ohne Beachtung der Groß- und Kleinschreibung – auseinander liegen, also wie man den einen String in den anderen durch ein- oder mehrmaliges Anwenden einer beliebigen der folgenden Aktionen überführen kann:

- Ein Zeichen hinzufügen ($+$),
- ein Zeichen löschen ($-$) oder
- ein Zeichen verändern ($\rightsquigarrow$).

Schreiben Sie eine Methode `int editDistance(String, String)`, die zeichenweise die drei Aktionen testet und den anderen Teil rekursiv prüft.

Beispiele Für die gezeigten Eingaben sind folgende Modifikationen nötig:

Eingabe 1	Eingabe 2	Resultat	Aktionen
"Micha"	"Michael"	2	$\text{Micha} \underbrace{\rightarrow}_{+\text{e}} \text{Michae} \underbrace{\rightarrow}_{+\text{l}} \text{Michael}$
"Ananas"	"Banane"	3	$\text{Ananas} \underbrace{\rightarrow}_{+\text{B}} \text{BAnanas} \underbrace{\rightarrow}_{-\text{s}} \text{BAnana} \underbrace{\rightarrow}_{\text{a}\rightsquigarrow\text{e}} \text{BAnane}$

Bonus (★★★☆☆) Optimieren Sie Edit Distance mit Memoization

8.3.3 Aufgabe 3: Longest Common Subsequence (★★★☆☆)

In der vorherigen Aufgabe ging es darum, wie viele Änderungen benötigt werden, um zwei gegebene Strings ineinander zu überführen. Eine weiteres interessantes Problem besteht darin, die längste gemeinsame, aber nicht unbedingt zusammenhängende Sequenz von Buchstaben zu ermitteln, die in zwei Strings in der gleichen Abfolge auftritt. Schreiben Sie eine Methode `String lcs(String, String)`, die rekursiv von hinten die Strings verarbeitet und bei zwei gleich langen Teilstücken das zweite nutzt.