Aufgabe 7: Dynamische Programmierung

Mittels Dynamischer Programmierung (auch Memoization genannt) kann man insbesondere rekursive Lösungen auf Kosten des Speicherbedarf beschleunigen, indem man Zwischenergebnisse „abspeichert“ und bei (wiederkehrendem) Bedarf „abruft“, ohne sie erneut berechnen zu müssen.

Gegeben sei folgende geschachtelt-rekursive Funktion für $n, m \geq 0$:

$$a(n,m) = \begin{cases} n + \lfloor \frac{n}{2} \rfloor & \text{falls } m = 0 \\ a(1, m-1), & \text{falls } n = 0 \wedge m \neq 0 \\ a(n + \lfloor \sqrt{a(n-1,m)} \rfloor, m-1), & \textit{sonst} \end{cases}$$

(a) Implementieren Sie die obige Funktion `a(n,m)` zunächst ohne weitere Optimierungen als Prozedur/Methode in einer Programmiersprache Ihrer Wahl.

```java
public long a(int n, int m) {
  if (m == 0) {
    return n + (n / 2);
  } else if (n == 0 && m != 0) {
    return a(1, m - 1);
  } else {
    return a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
  }
}
```

github: raw

(b) Geben Sie nun eine DP-Implementierung der Funktion `a(n,m)` an, die `a(n,m)` für $0 \geq n \geq 100000$ und $0 \geq m \geq 25$ höchstens einmal gemäß obiger rekursiver Definition berechnet. Beachten Sie, dass Ihre Prozedur trotzdem auch weiterhin mit $n > 100000$ und $m > 25$ aufgerufen werden können soll.

```java
long[][] tmp = new long[100001][26];
// mit For - Schleife durchiterieren und anfangs mit -1 fuellen

public long aDp(int n, int m) {
  if (n <= 100000 && m <= 25 && tmp[n][m] != -1) {
    return tmp[n][m];
  } else {
    long merker;
    if (m == 0) {
      merker = n + (n / 2);
    } else if (n == 0 && m != 0) {
      merker = a(1, m - 1);
    } else {
      merker = a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
    }
    if (n <= 100000 && m <= 25) {
      tmp[n][m] = merker;
    }
    return merker;
  }
```

```
34      }
```

github: raw