

Aufgabe 1: „Rekursion und Induktion“

- (a) Gegeben sei die Methode `BigInteger lfBig(int n)` zur Berechnung der eingeschränkten Linksfakultät:

$$!n := \begin{cases} n!(n-1) - (n-1)!(n-2) & \text{falls } 1 < n < 32767 \\ 1 & \text{falls } n = 1 \\ 0 & \text{sonst} \end{cases}$$

```

3  import java.math.BigInteger;
4  import static java.math.BigInteger.ZERO;
5  import static java.math.BigInteger.ONE;
6
7  public class LeftFactorial {
8
9      BigInteger sub(BigInteger a, BigInteger b) {
10         return a.subtract(b);
11     }
12
13     BigInteger mul(BigInteger a, BigInteger b) {
14         return a.multiply(b);
15     }
16
17     BigInteger mul(int a, BigInteger b) {
18         return mul(BigInteger.valueOf(a), b);
19     }
20
21     // returns the left factorial !n
22     BigInteger lfBig(int n) {
23         if (n <= 0 || n >= Short.MAX_VALUE) {
24             return ZERO;
25         } else if (n == 1) {
26             return ONE;
27         } else {
28             return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
29         }
30     }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

Implementieren Sie unter Verwendung des Konzeptes der *dynamischen Programmierung* die Methode `BigInteger dp(int n)`, die jede $!n$ auch bei mehrfachem Aufrufen mit dem gleichen Parameter höchstens einmal rekursiv berechnet. Sie dürfen der Klasse `LeftFactorial` genau ein Attribut beliebigen Datentyps hinzufügen und die in `lfBig(int)` verwendeten Methoden und Konstanten ebenfalls nutzen.

Wir führen ein Attribut mit dem Namen `store` ein und erzeugen ein Feld vom Typ `BigInteger` mit der Länge $n + 1$. Die Länge des Feld $n + 1$ hat den Vorteil, dass nicht ständig $n - 1$ verwendet werden muss, um den gewünschten Wert zu erhalten.

In der untenstehenden Implementation gibt es zwei Methoden mit dem Namen `dp`. Die untenstehende Methode ist nur eine Hüllmethode, mit der nach außen hin die Berechnung gestartet und das `store`-Feld neu gesetzt wird. So ist es möglich `dp()` mehrmals hintereinan-

der mit verschiedenen Werten aufzurufen (siehe `main()`-Methode).

```

32  BigInteger[] store;
33
34  BigInteger dp(int n, BigInteger[] store) {
35      if (n > 1 && store[n] != null) {
36          return store[n];
37      }
38      if (n <= 0 || n >= Short.MAX_VALUE) {
39          return ZERO;
40      } else if (n == 1) {
41          return ONE;
42      } else {
43          BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1,
44              ↪ dp(n - 2, store)));
45          store[n] = result;
46          return result;
47      }
48
49  BigInteger dp(int n) {
50      store = new BigInteger[n + 1];
51      return dp(n, store);
52  }

```

Code-Beispiel auf Github ansehen:
[src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

- (b) Betrachten Sie nun die Methode `lfLong(int)` zur Berechnung der vorangehend definierten Links fakultät ohne obere Schranke. Nehmen Sie im Folgenden an, dass der Datentyp `long` unbeschränkt ist und daher kein Überlauf auftritt.

```

54  long lfLong(int n) {
55      if (n <= 0) {
56          return 0;
57      } else if (n == 1) {
58          return 1;
59      } else {
60          return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
61      }
62  }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)

Beweisen Sie *formal* mittels *vollständiger Induktion*:

$$\forall n \geq 0 : \text{lfLong}(n) \equiv \sum_{k=0}^{n-1} k!$$

Induktionsanfang — Beweise, dass $A(1)$ eine wahre Aussage ist. —

$$n = 1 \Rightarrow \text{lfLong}(1) = 1 = \sum_{k=0}^{n-1} k! = 0! = 1$$

$$\begin{aligned}
n = 2 &\Rightarrow \text{lfLong}(2) \\
&= (n + 1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! \\
&= 2 * \text{lfLong}(1) - 1 * \text{lfLong}(0) \\
&= 2 \\
&= \sum_{k=0}^1 k! \\
&= 1! + 0! \\
&= 1 + 1 \\
&= 2
\end{aligned}$$

Induktionsvoraussetzung — Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. _____

$$\text{lfLong}(n) = \sum_{k=0}^{n-1} k!$$

Induktionsschritt — Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss. _____

$$\begin{aligned}
A(n+1) &= \text{lfLong}(n+1) \\
&= (n+1) * \text{lfLong}(n) - n * \text{lfLong}(n-1) \\
&= (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{(n-1)-1} k! && \text{Formel eingesetzt} \\
&= (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{subtrahiert} \\
&= n \sum_{k=0}^{n-1} k! + \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{ausmultipliziert mit } (n+1) \\
&= \sum_{k=0}^{n-1} k! + n \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{Reihenfolge der Terme ändern} \\
&= \sum_{k=0}^{n-1} k! + n \left((n-1)! + \sum_{k=0}^{n-2} k! \right) - n \sum_{k=0}^{n-2} k! && (n-1)! \text{ aus Summenzeichen entfernen} \\
&= \sum_{k=0}^{n-1} k! + n \left((n-1)! + \sum_{k=0}^{n-2} k! - \sum_{k=0}^{n-2} k! \right) && \text{Distributivgesetz } ac - bc = (a-b)c \\
&= \sum_{k=0}^{n-1} k! + n(n-1)! && +\Sigma - \Sigma = 0 \\
&= \sum_{k=0}^{n-1} k! + n! && \text{Fakultät erhöht} \\
&= \sum_{k=0}^n k! && \text{Element zum Summenzeichen hinzugefügt} \\
&= \sum_{k=0}^{(n+1)-1} k! && \text{mit } (n+1) \text{ an der Stelle von } n
\end{aligned}$$

Additum Komplette Klasse LeftFactorial

```

3 import java.math.BigInteger;
4 import static java.math.BigInteger.ZERO;
5 import static java.math.BigInteger.ONE;
6
7 public class LeftFactorial {
8
9     BigInteger sub(BigInteger a, BigInteger b) {
10         return a.subtract(b);
11     }
12
13     BigInteger mul(BigInteger a, BigInteger b) {

```

```
14     return a.multiply(b);
15 }
16
17 BigInteger mul(int a, BigInteger b) {
18     return mul(BigInteger.valueOf(a), b);
19 }
20
21 // returns the left factorial !n
22 BigInteger lfBig(int n) {
23     if (n <= 0 || n >= Short.MAX_VALUE) {
24         return ZERO;
25     } else if (n == 1) {
26         return ONE;
27     } else {
28         return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
29     }
30 }
31
32 BigInteger[] store;
33
34 BigInteger dp(int n, BigInteger[] store) {
35     if (n > 1 && store[n] != null) {
36         return store[n];
37     }
38     if (n <= 0 || n >= Short.MAX_VALUE) {
39         return ZERO;
40     } else if (n == 1) {
41         return ONE;
42     } else {
43         BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1,
44             ↪ dp(n - 2, store)));
45         store[n] = result;
46         return result;
47     }
48 }
49
50 BigInteger dp(int n) {
51     store = new BigInteger[n + 1];
52     return dp(n, store);
53 }
54
55 long lfLong(int n) {
56     if (n <= 0) {
57         return 0;
58     } else if (n == 1) {
59         return 1;
60     } else {
61         return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
62     }
63 }
64
65 public static void main(String[] args) {
66     LeftFactorial lf = new LeftFactorial();
67
68     for (int i = 0; i < 15; i++) {
69         System.out.println(lf.lfBig(i));
70     }
71
72     for (int i = 0; i < 15; i++) {
73         System.out.println(lf.dp(i));
74     }
75 }
```

```
74
75     for (int i = 0; i < 15; i++) {
76         System.out.println(1f.lfLong(i));
77     }
78 }
79 }
```

Code-Beispiel auf Github ansehen:
[src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java](https://github.com/orgs/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java)