Einzelprüfung „Theoretische Informatik / Algorithmen (vertieft)“

Einzelprüfungsnummer 66115 / 2014 / Frühjahr

Thema 1 / Aufgabe 1

(Klasse „LeftFactorial“ und Methode „lfBig()“)

Stichwörter: Vollständige Induktion, Rekursion, Implementierung in Java, Dynamische Programmierung

---

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

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

```java
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
import static java.math.BigInteger.ONE;

public class LeftFactorial {

  BigInteger sub(BigInteger a, BigInteger b) {
    return a.subtract(b);
  }

  BigInteger mul(BigInteger a, BigInteger b) {
    return a.multiply(b);
  }

  BigInteger mul(int a, BigInteger b) {
    return mul(BigInteger.valueOf(a), b);
  }

  // returns the left factorial !n
  BigInteger lfBig(int n) {
    if (n <= 0 || n >= Short.MAX_VALUE) {
      return ZERO;
    } else if (n == 1) {
      return ONE;
    } else {
      return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
    }
  }
```

Code-Beispiel auf Github ansehen: `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.

Lösungsvorschlag

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

```
BigInteger[] store;

BigInteger dp(int n, BigInteger[] store) {
  if (n > 1 && store[n] != null) {
    return store[n];
  }
  if (n <= 0 || n >= Short.MAX_VALUE) {
    return ZERO;
  } else if (n == 1) {
    return ONE;
  } else {
    BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2,
    ↪  store)));
    store[n] = result;
    return result;
  }
}

BigInteger dp(int n) {
  store = new BigInteger[n + 1];
  return dp(n, store);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

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

```
long lfLong(int n) {
  if (n <= 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

Beweisen Sie *formal* mittels *vollständiger Induktion*:

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

Lösungsvorschlag

Induktionsanfang

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

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

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

Induktionsvoraussetzung

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

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

Induktionsschritt

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

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

Additum

Komplette Klasse LeftFactorial

```
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
import static java.math.BigInteger.ONE;

public class LeftFactorial {

  BigInteger sub(BigInteger a, BigInteger b) {
```

```
  return a.subtract(b);
}

BigInteger mul(BigInteger a, BigInteger b) {
  return a.multiply(b);
}

BigInteger mul(int a, BigInteger b) {
  return mul(BigInteger.valueOf(a), b);
}

// returns the left factorial !n
BigInteger lfBig(int n) {
  if (n <= 0 || n >= Short.MAX_VALUE) {
    return ZERO;
  } else if (n == 1) {
    return ONE;
  } else {
    return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
  }
}

BigInteger[] store;

BigInteger dp(int n, BigInteger[] store) {
  if (n > 1 && store[n] != null) {
    return store[n];
  }
  if (n <= 0 || n >= Short.MAX_VALUE) {
    return ZERO;
  } else if (n == 1) {
    return ONE;
  } else {
    BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2,
    ↪  store)));
    store[n] = result;
    return result;
  }
}

BigInteger dp(int n) {
  store = new BigInteger[n + 1];
  return dp(n, store);
}

long lfLong(int n) {
  if (n <= 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
  }
}
```

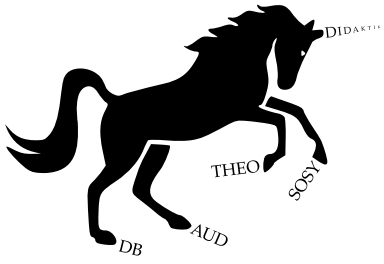
```
  public static void main(String[] args) {
    LeftFactorial lf = new LeftFactorial();

    for (int i = 0; i < 15; i++) {
      System.out.println(lf.lfBig(i));
    }

    for (int i = 0; i < 15; i++) {
      System.out.println(lf.dp(i));
    }

    for (int i = 0; i < 15; i++) {
      System.out.println(lf.lfLong(i));
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Hilf mit! Die Hermine schafft das nicht alleine! Das ist ein Community-Projekt. Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieses Dokuments kann unter folgender URL aufgerufen werden: `https://github.com/hbschlang/lehramt-informatik/blob/main/Staatsexamen/66115/2014/03/Thema-1/Aufgabe-1.tex`