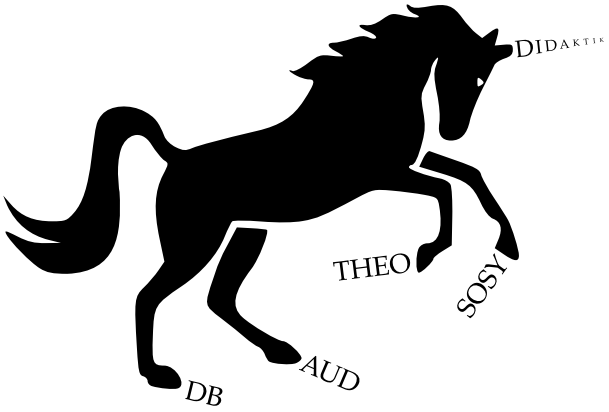
66115
Frühjahr
2014

Theoretische Informatik / Algorithmen (vertieft)

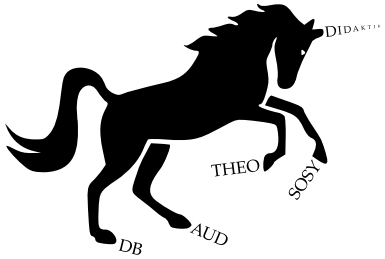
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



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.

Thema Nr. 1

Aufgabe 1: „Rekursion und Induktion“ [Klasse „LeftFactorial“ und Methode „lfBig()“]

(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.

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. ——

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

Aufgabe 2 [Binäre Bäume]

Implementieren Sie in einer objekt-orientierten Sprache Ihrer Wahl eine Klasse namens `BinBaum`, deren Instanzen binäre Bäume mit ganzzahligen Datenknoten darstellen, nach folgender Spezifikation:

(a) Beginnen Sie zunächst mit der Datenstruktur selbst:

- Mit Hilfe des Konstruktors soll ein neuer Baum erstellt werden, der aus einem einzelnen Knoten besteht, in dem der dem Konstruktor als Parameter übergebene Wert (ganzzahlig, in Java z.B. `int`) gespeichert ist.

Lösungsvorschlag

```java
class Knoten {
  int value;

  Knoten left;
  Knoten right;

  public Knoten(int value) {
    this.value = value;
  }
```

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

- Die Methoden `setLeft(int value)` bzw. `setRight(int value)` sollen den linken bzw. rechten Teilbaum des aktuellen Knotens durch einen jeweils neuen Teilbaum ersetzen, der seinerseits aus einem einzelnen Knoten besteht, in dem der übergebene Wert `value` gespeichert ist. Sie haben keinen Rückgabewert.

Lösungsvorschlag

```java
  public void setLeft(Knoten left) {
    this.left = left;
  }

  public void setRight(Knoten right) {
    this.right = right;
  }
```

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

- Die Methoden `getLeft()` bzw. `getRight()` sollen den linken bzw. rechten Teilbaum zurückgeben (bzw. `null`, wenn keiner vorhanden ist)

Lösungsvorschlag

```java
  public Knoten getLeft() {
    return left;
  }

  public Knoten getRight() {
    return right;
  }
```

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

- Die Methode `int getValue()` soll den Wert zurückgeben, der im aktuellen Wurzelknoten gespeichert ist.

Lösungsvorschlag

```java
  public int getValue() {
    return value;
  }
```

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

(b) Implementieren Sie nun die Methoden `preOrder()` bzw. `postOrder()`. Sie sollen die Knoten des Baumes mit Tiefensuche traversieren, die Werte dabei in pre-order bzw. post-order Reihenfolge in eine Liste (z.B. `List<Integer>`) speichern und diese Ergebnisliste zurückgeben. Die Tiefensuche soll dabei zuerst in den linken und dann in den rechten Teilbaum absteigen.

Lösungsvorschlag

```
void preOrder(Knoten knoten, List<Integer> list) {
  if (knoten != null) {
    list.add(knoten.getValue());
    preOrder(knoten.getLeft(), list);
    preOrder(knoten.getRight(), list);
  }
}

List<Integer> preOrder() {
  List<Integer> list = new ArrayList<>();
  preOrder(head, list);
  return list;
}

void postOrder(Knoten knoten, List<Integer> list) {
  if (knoten != null) {
    postOrder(knoten.getLeft(), list);
    postOrder(knoten.getRight(), list);
    list.add(knoten.getValue());
  }
}

List<Integer> postOrder() {
  List<Integer> list = new ArrayList<>();
  postOrder(head, list);
  return list;
}
```

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

(c) Ergänzen Sie schließlich die Methode `isSearchTree()`. Sie soll überprüfen, ob der Binärbaum die Eigenschaften eines binären Suchbaums erfüllt. Beachten Sie, dass die Laufzeit-Komplexität Ihrer Implementierung linear zur Anzahl der Knoten im Baum sein muss.

Lösungsvorschlag

```
boolean isSearchTree(Knoten knoten) {
  if (knoten == null) {
    return true;
  }

  if (knoten.getLeft() != null && knoten.getValue() <
↪ knoten.getLeft().getValue()) {
    return false;
  }
```

```java
    if (knoten.getRight() != null && knoten.getValue() >
↪   knoten.getRight().getValue()) {
      return false;
    }

    return isSearchTree(knoten.getLeft()) && isSearchTree(knoten.getRight());
  }
```

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

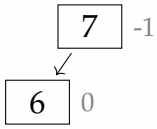
Aufgabe 3 [Einfügen und dreimal einen Knoten löschen]

(a) Fügen Sie die Zahlen (7, 6, 2, 1, 5, 3, 8, 4) in dieser Reihenfolge in einen anfangs leeren AVL Baum ein. Stellen Sie die AVL Eigenschaft ggf. nach jedem Einfügen mit geeigneten Rotationen wieder her. Zeichnen Sie den AVL Baum einmal vor und einmal nach jeder einzelnen Rotation.

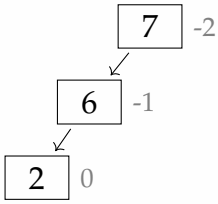
Lösungsvorschlag

Nach dem Einfügen von „7“:

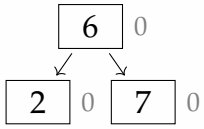
Nach dem Einfügen von „6“:



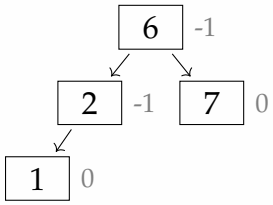
Nach dem Einfügen von „2“:



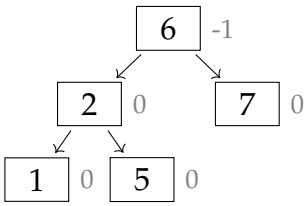
Nach der Rechtsrotation:



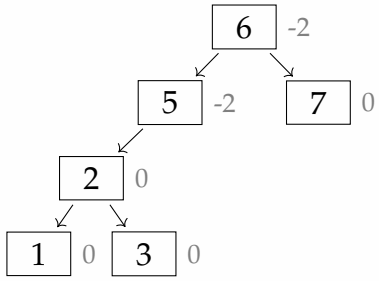
Nach dem Einfügen von „1“:



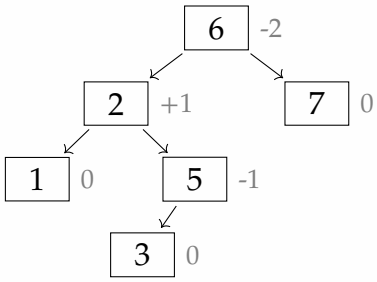
Nach dem Einfügen von „5“:



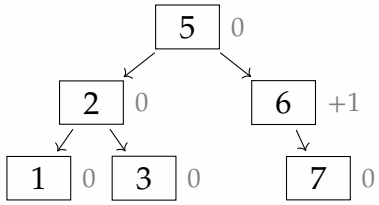
Nach der Linksrotation:



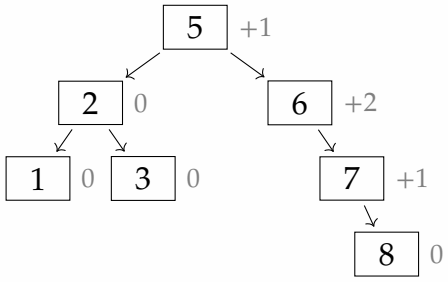
Nach dem Einfügen von „3“:



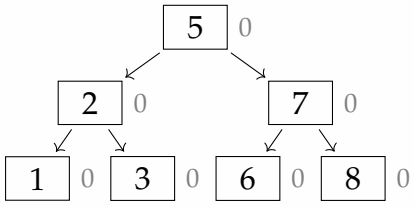
Nach der Rechtsrotation:



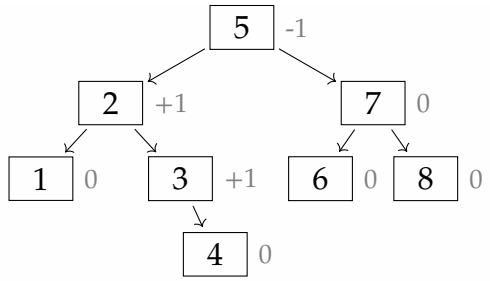
Nach dem Einfügen von „8“:



Nach der Linksrotation:

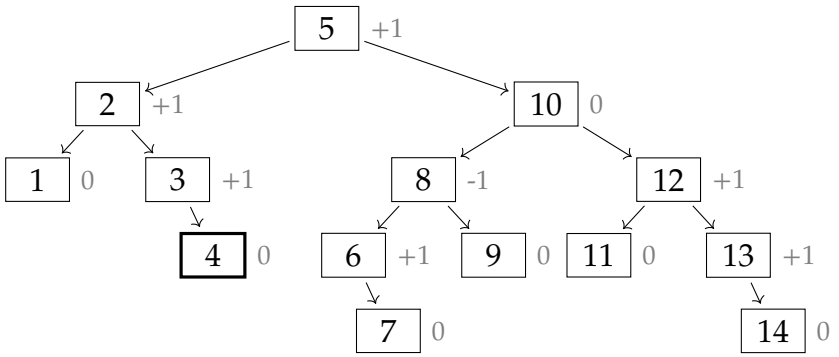


Nach dem Einfügen von „4“:



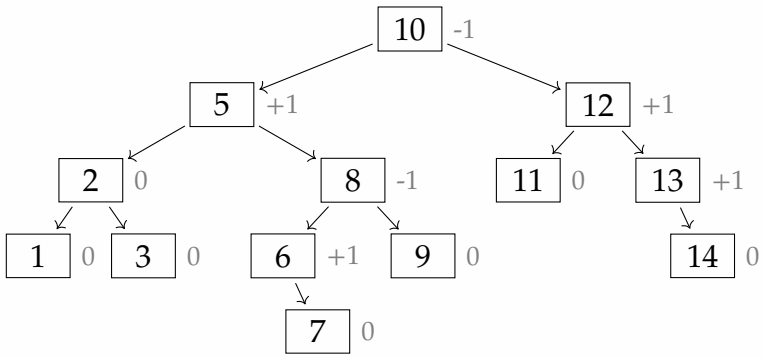
(b) Entfernen Sie den jeweils markierten Knoten aus den folgenden AVL-Bäumen. Stellen Sie die AVL-Eigenschaft ggf. durch geeignete Rotationen wieder her. Zeichnen Sie nur den resultierenden Baum.

(i) Baum 1:

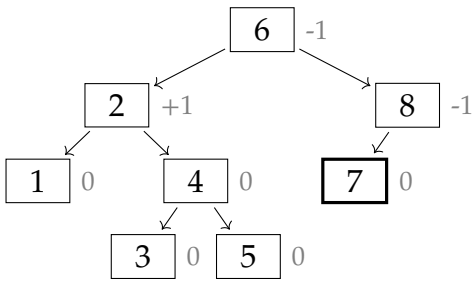


Lösungsvorschlag

Nach dem Löschen von „4“:

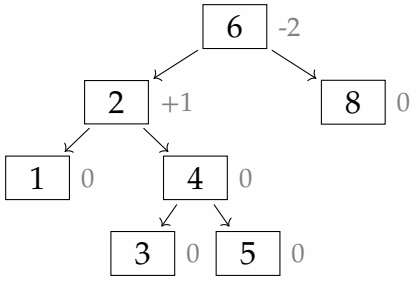


(ii) Baum 2:

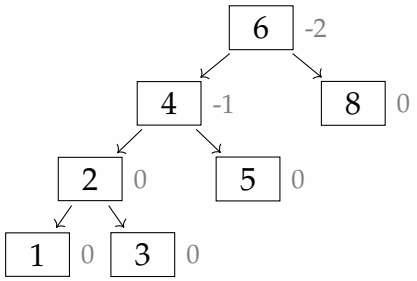


Lösungsvorschlag

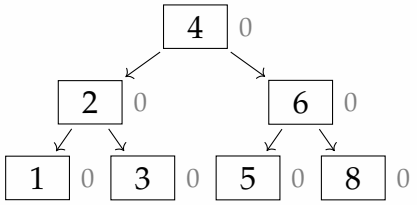
Nach dem Löschen von „7“:



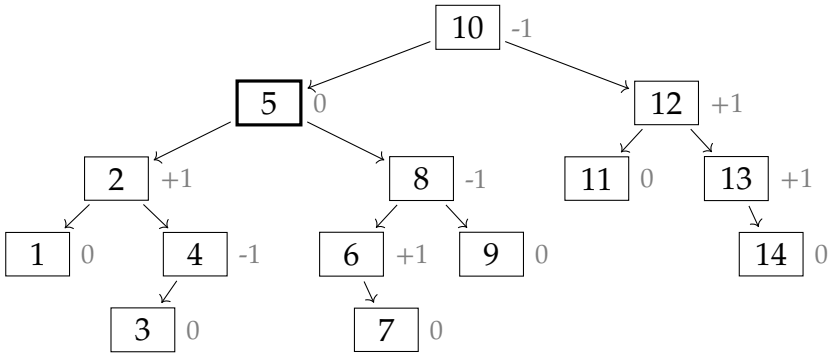
Nach der Linksrotation:



Nach der Rechtsrotation:

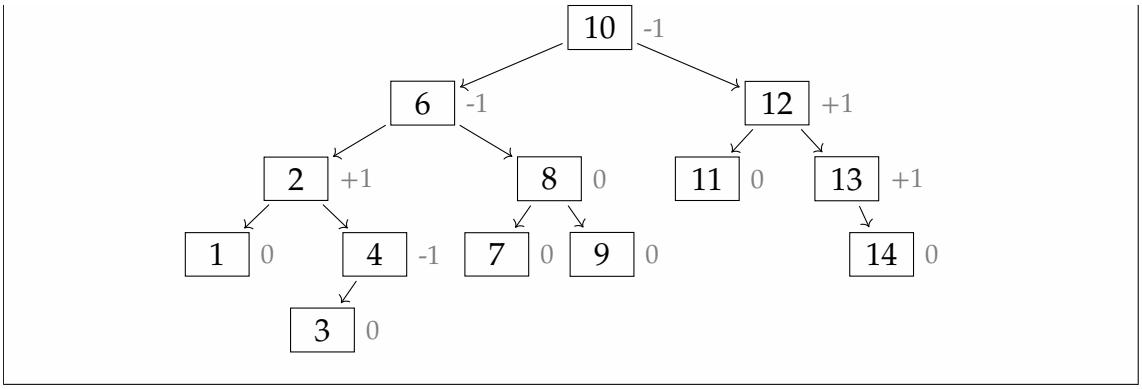


(iii) Baum 3:



Lösungsvorschlag

Nach dem Löschen von „5“:



Aufgabe 5 [Halteproblem H m]

(a) Definieren Sie die zum Halteproblem für Turing-Maschinen bei fester Eingabe $m \in \mathbb{N}_0 = \{0, 1, 2, \dots\}$ gehörende Menge H_m.

Lösungsvorschlag

$H_m = \{\, c(M) \in \mathbb{N} \mid c(M)$ hält auf Eingabe $m \,\}$, wobei $c(M)$ der Codierung der Turingmaschine (Gödelnummer) entspricht.

(b) Gegeben sei das folgende Problem E:

Entscheiden Sie, ob es für die deterministische Turing-Maschine mit der Gödelnummer n eine Eingabe $w \in \mathbb{N}_0$ gibt, so dass w eine gerade Zahl ist und die Maschine n gestartet mit w hält.

Zeigen. Sie, dass E nicht entscheidbar ist. Benutzen Sie, dass H_m aus (a) für jedes $m \in \mathbb{N}_0$ nicht entscheidbar ist.

Lösungsvorschlag

Wir zeigen dies durch Reduktion $H_2 \leq E$:

- Berechenbare Funktion f: lösche Eingabe, schreibe eine 2 und starte dich selbst.
- M ist eine Turingmaschine, die E entscheidet.
- $x \in H_2$ (Quellcode der Programme, die auf die Eingabe von 2 halten)
- M_x (kompiliertes Programm, TM)
- Für alle $x \in H_2$ gilt, M_x hält auf Eingabe von 2 $\Leftrightarrow f(x) = c(M) \in E$. Denn sofern die ursprüngliche Maschine auf das Wort 2 hält, hält M auf alle Eingaben und somit auch auf Eingaben gerader Zahlen. Hält die ursprüngliche Maschine M nicht auf die Eingabe der Zahl 2, so hält M auf keine Eingabe.

(c) Zeigen Sie, dass das Problem E aus (b) partiell-entscheidbar (= rekursiv aufzählbar) ist.