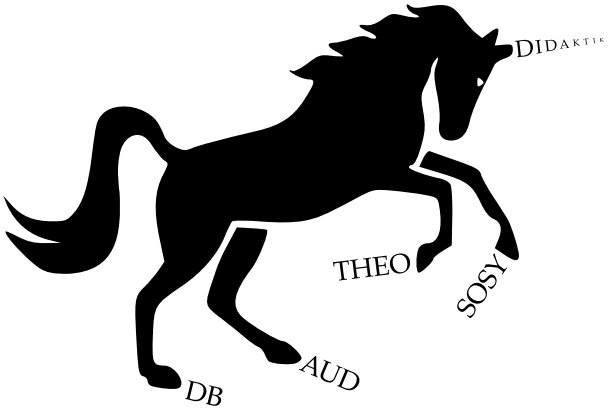
46115
Frühjahr
2014

Theoretische Informatik / Algorithmen / Datenstrukturen (nicht vertieft)

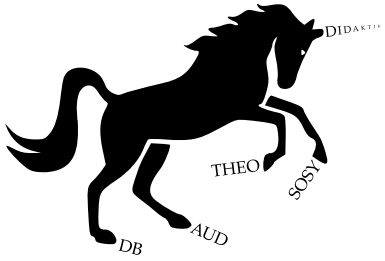
Aufgabenstellungen mit Lösungsvorschlägen



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.

Diese Materialsammlung unterliegt den Bestimmungen der Creative Commons Namensnennung-Nicht kommerziell-Share Alike 4.0 International-Lizenz.

Thema Nr. 1

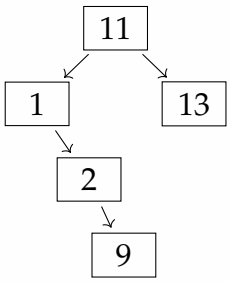
Aufgabe 7 [Zahlenfolge in binärer Suchbaum, Min-Heap und AVL-Baum]

Fügen Sie nacheinander die Zahlen 11, 1, 2, 13, 9, 10, 7, 5

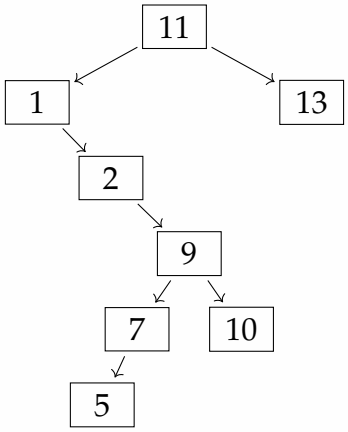
(a) in einen leeren binären Suchbaum und zeichnen Sie den Suchbaum jeweils nach dem Einfügen von „9“ und „5“

Lösungsvorschlag

Nach dem Einfügen von „9“:



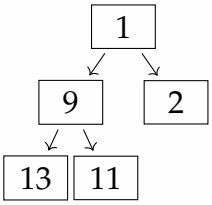
Nach dem Einfügen von „5“:



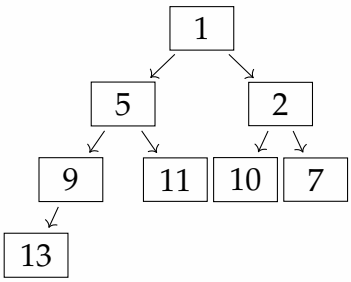
(b) in einen leeren Min-Heap ein, der bzgl. „$\leq$“ angeordnet ist und geben Sie den Heap nach „9“ und nach „5“ an

Lösungsvorschlag

Nach dem Einfügen von „9“:



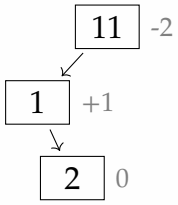
Nach dem Einfügen von „5“:



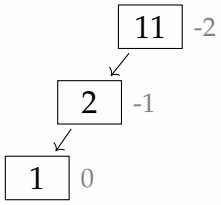
(c) in einen leeren AVL-Baum ein! Geben Sie den AVL Baum nach „2“ und „5“ an und beschreiben Sie die ggf. notwendigen Rotationen beim Einfügen dieser beiden Elemente!

Lösungsvorschlag

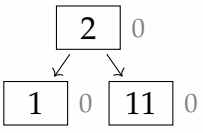
Nach dem Einfügen von „2“:



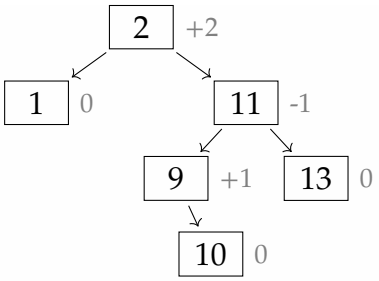
Nach der Linksrotation:



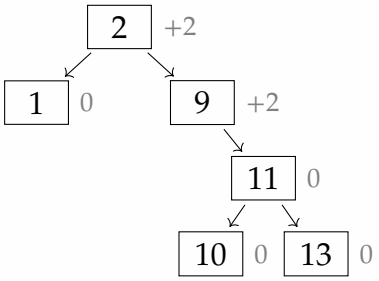
Nach der Rechtsrotation:



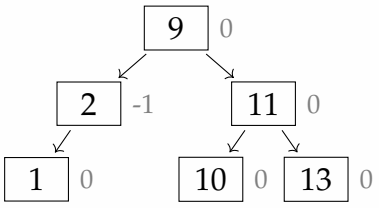
Nach dem Einfügen von „10“:



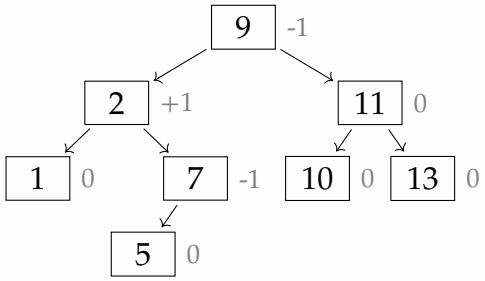
Nach der Rechtsrotation:



Nach der Linksrotation:



Nach dem Einfügen von „5“:

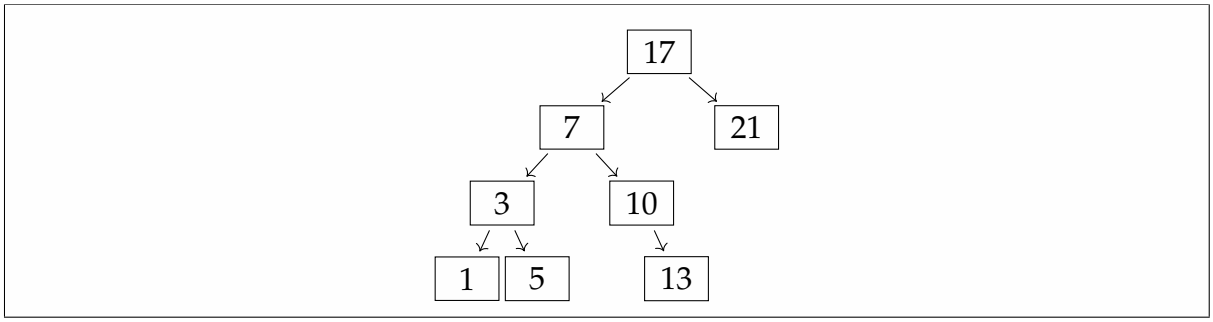


Thema Nr. 2

Frühjahr 2014 (46115) - Thema 2 Aufgabe 3 [Binärer Suchbaum 17, 7, 21, 3, 10, 13, 1, 5]

(a) Fügen Sie die Zahlen 17, 7, 21, 3, 10, 13, 1, 5 nacheinander in der vorgegebenen Reihenfolge in einen binären Suchbaum ein und zeichnen Sie das Ergebnis!

Lösungsvorschlag



(b) Implementieren Sie in einer objektorientierten Programmiersprache eine rekursiv festgelegte Datenstruktur, deren Gestaltung sich an folgender Definition eines binären Baumes orientiert!

Ein binärer Baum ist entweder ein leerer Baum oder besteht aus einem Wurzelelement, das einen binären Baum als linken und einen als rechten Teilbaum besitzt. Bei dieser Teilaufgabe können Sie auf die Implementierung von Methoden (außer ggf. notwendigen Konstruktoren) verzichten!

Klasse „Knoten“

```
class Knoten {
  public int wert;
  public Knoten links;
  public Knoten rechts;
  public Knoten elternKnoten;

  Knoten(int wert) {
    this.wert = wert;
    links = null;
    rechts = null;
    elternKnoten = null;
  }

  public Knoten findeMiniumRechterTeilbaum() {
  }

  public void anhängen (Knoten knoten) {
  }
}
```

Klasse „BinärerSuchbaum“

```
public class BinärerSuchbaum {
  public Knoten wurzel;

  BinärerSuchbaum(Knoten wurzel) {
    this.wurzel = wurzel;
  }

  BinärerSuchbaum() {
    this.wurzel = null;
  }

  public void einfügen(Knoten knoten) {
  }

  public void einfügen(Knoten knoten, Knoten elternKnoten) {
  }

  public Knoten suchen(int wert) {
  }

  public Knoten suchen(int wert, Knoten knoten) {
  }

}
```

(c) Beschreiben Sie durch Implementierung in einer gängigen objektorientierten Programmiersprache, wie bei Verwendung der obigen Datenstruktur die Methode `loescheKnoten(w)` gestaltet sein muss, mit der der Knoten mit dem Eintrag `w` aus dem Baum entfernt werden kann, ohne die Suchbaumeigenschaft zu verletzen!

Lösungsvorschlag

```
  public void loescheKnoten(int w) {
    Knoten knoten = suchen(w);
    if (knoten == null) return;
    // Der Knoten hat keine Teilbäume.
    if (knoten.links == null && knoten.rechts == null) {
      if (w < knoten.elternKnoten.wert) {
        knoten.elternKnoten.links = null;
      } else {
        knoten.elternKnoten.rechts = null;
      }
    }

    // Der Knoten besitzt einen Teilbaum.
    // links
    else if (knoten.links != null && knoten.rechts == null) {
      knoten.elternKnoten.anhängen(knoten.links);
    }
    // rechts
    else if (knoten.links == null) {
      knoten.elternKnoten.anhängen(knoten.rechts);
    }
```

```
      // Der Knoten besitzt zwei Teilbäume.
      else {
        Knoten minimumKnoten = knoten.findeMiniumRechterTeilbaum();
        minimumKnoten.links = knoten.links;
        minimumKnoten.rechts = knoten.rechts;
        knoten.elternKnoten.anhängen(minimumKnoten);
      }
    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/BinaererSuchbaum.java`

Lösungsvorschlag

Klasse „BinärerSuchbaum“

```
public class BinaererSuchbaum {
  public Knoten wurzel;

  BinaererSuchbaum(Knoten wurzel) {
    this.wurzel = wurzel;
  }

  BinaererSuchbaum() {
    this.wurzel = null;
  }

  public void einfügen(Knoten knoten) {
    if (wurzel != null) {
      einfügen(knoten, wurzel);
    } else {
      wurzel = knoten;
      knoten.elternKnoten = wurzel;
    }
  }

  public void einfügen(int wert) {
    einfügen(new Knoten(wert));
  }

  public void einfügen(Knoten knoten, Knoten elternKnoten) {
    if (knoten.wert <= elternKnoten.wert) {
      if (elternKnoten.links != null) {
        einfügen(knoten, elternKnoten.links);
      } else {
        elternKnoten.links = knoten;
        knoten.elternKnoten = elternKnoten;
      }
    } else {
      if (elternKnoten.rechts != null) {
        einfügen(knoten, elternKnoten.rechts);
      } else {
        elternKnoten.rechts = knoten;
        knoten.elternKnoten = elternKnoten;
      }
```

```
    }
  }

  public Knoten suchen(int wert) {
    if (wurzel == null || wurzel.wert == wert) {
      return wurzel;
    } else {
      return suchen(wert, wurzel);
    }
  }

  public Knoten suchen(int wert, Knoten knoten) {
    if (knoten.wert == wert) {
      return knoten;
    } else if (wert < knoten.wert && knoten.links != null) {
      return suchen(wert, knoten.links);
    } else if (wert > knoten.wert && knoten.rechts != null) {
      return suchen(wert, knoten.rechts);
    }
    return null;
  }

  public void loescheKnoten(int w) {
    Knoten knoten = suchen(w);
    if (knoten == null) return;
    // Der Knoten hat keine Teilbäume.
    if (knoten.links == null && knoten.rechts == null) {
      if (w < knoten.elternKnoten.wert) {
        knoten.elternKnoten.links = null;
      } else {
        knoten.elternKnoten.rechts = null;
      }
    }

    // Der Knoten besitzt einen Teilbaum.
    // links
    else if (knoten.links != null && knoten.rechts == null) {
      knoten.elternKnoten.anhängen(knoten.links);
    }
    // rechts
    else if (knoten.links == null) {
      knoten.elternKnoten.anhängen(knoten.rechts);
    }

    // Der Knoten besitzt zwei Teilbäume.
    else {
      Knoten minimumKnoten = knoten.findeMiniumRechterTeilbaum();
      minimumKnoten.links = knoten.links;
      minimumKnoten.rechts = knoten.rechts;
      knoten.elternKnoten.anhängen(minimumKnoten);
    }
  }

  // Der Baum aus dem Foliensatz
```

```
  public BinaererSuchbaum erzeugeTestBaum() {
    BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
    binärerSuchbaum.einfügen(new Knoten(7));
    binärerSuchbaum.einfügen(new Knoten(3));
    binärerSuchbaum.einfügen(new Knoten(11));
    binärerSuchbaum.einfügen(new Knoten(2));
    binärerSuchbaum.einfügen(new Knoten(6));
    binärerSuchbaum.einfügen(new Knoten(9));
    binärerSuchbaum.einfügen(new Knoten(1));
    binärerSuchbaum.einfügen(new Knoten(5));
    return binärerSuchbaum;
  }

  public void ausgebenInOrder() {

  }

  public static void main(String[] args) {
    BinaererSuchbaum binärerSuchbaum = new BinaererSuchbaum();
    BinaererSuchbaum testBaum = binärerSuchbaum.erzeugeTestBaum();

    // Teste das Einfügen

    System.out.println(testBaum.wurzel.wert); // 7
    System.out.println(testBaum.wurzel.links.wert); // 3
    System.out.println(testBaum.wurzel.links.links.wert); // 2
    System.out.println(testBaum.wurzel.links.rechts.wert); // 6
    System.out.println(testBaum.wurzel.rechts.wert); // 11

    // Teste das Suchen

    System.out.println("Gesucht nach 5 und gefunden: " + testBaum.suchen(5).wert);
    System.out.println("Gesucht nach 9 und gefunden: " + testBaum.suchen(9).wert);
    System.out.println("Gesucht nach 7 und gefunden: " + testBaum.suchen(7).wert);
    System.out.println("Gesucht nach 10 und gefunden: " + testBaum.suchen(10));

    // Teste das Löschen

    // Der Knoten hat keine Teilbäume.
    System.out.println("Noch nicht gelöschter Knoten 9: " + testBaum.suchen(9).wert);
    testBaum.loescheKnoten(9);
    System.out.println("Gelöschter Knoten 9: " + testBaum.suchen(9));

    // Der Knoten hat einen Teilbaum.
    // fristen Testbaum erzeugen.
    testBaum = binärerSuchbaum.erzeugeTestBaum();
    Knoten elternKnoten = testBaum.suchen(3);
    System.out.println("Rechts Kind von 3 vor dem Löschen: " +
↪   elternKnoten.rechts.wert);
    testBaum.loescheKnoten(6);
    System.out.println("Rechts Kind von 3Nach dem Löschen: " +
↪   elternKnoten.rechts.wert);

    // Der Knoten hat zwei Teilbäume.
```

```
    // fristen Testbaum erzeugen.
    testBaum = binärerSuchbaum.erzeugeTestBaum();
    Knoten wurzel = testBaum.wurzel;
    System.out.println("Linkes Kind der Wurzel vor dem Löschen: " + wurzel.links.wert);
↪   // 5
    testBaum.loescheKnoten(3);
    System.out.println("Linkes Kind der WurzelNach dem Löschen: " + wurzel.links.wert);
↪   // 3
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/BinaererSuchbaum.java

Klasse „Knoten“

```
class Knoten {
  public int wert;
  public Knoten links;
  public Knoten rechts;
  public Knoten elternKnoten;

  Knoten(int wert) {
    this.wert = wert;
    links = null;
    rechts = null;
    elternKnoten = null;
  }

  public Knoten findeMiniumRechterTeilbaum() {
    if (rechts != null) {
      Knoten minimumKonten = rechts;
      while (minimumKonten.links != null) {
        minimumKonten = minimumKonten.links;
      }
      return minimumKonten;
    }
    return null;
  }

  public void anhängen (Knoten knoten) {
    if (knoten.wert < wert) {
      links = knoten;
    } else {
      rechts = knoten;
    }
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/suchbaum/Knoten.java

Aufgabe 4 [Binomialkoeffizient]

Aufgabe 4

Für Binomialkoeffizienten $\binom{n}{k}$ gelten neben den grundlegenden Beziehungen $\binom{n}{0} = 1$ und $\binom{n}{n} = 1$ auch folgende Formeln:

> **Exkurs: Binomialkoeffizient**
> Der Binomialkoeffizient ist eine mathematische Funktion, mit der sich eine der Grundaufgaben der Kombinatorik lösen lässt. Er gibt an, auf wie viele verschiedene Arten man k bestimmte Objekte aus einer Menge von n verschiedenen Objekten auswählen kann (ohne Zurücklegen, ohne Beachtung der Reihenfolge). Der Binomialkoeffizient ist also die Anzahl der k-elementigen Teilmengen einer n-elementigen Menge. [a]
>
> [a] `https://de.wikipedia.org/wiki/Binomialkoeffizient`

A $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$

B $\binom{n}{k} = \binom{n-1}{k-1} \cdot \frac{n}{k}$

(a) Implementieren Sie unter Verwendung von Beziehung (A) eine rekursive Methode `binRek(n, k)` zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!

Lösungsvorschlag

Zuerst verwandeln wir die Beziehung (A) geringfügig um, indem wir n durch $n-1$ ersetzen:

$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

```java
public static int binRek(int n, int k) {
  if (k == 0 || k == n) {
    return 1;
  } else {
    return binRek(n - 1, k - 1) + binRek(n - 1, k);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java`

(b) Implementieren Sie unter Verwendung von Beziehung (B) eine iterative Methode `binIt(n, k)` zur Berechnung des Binomialkoeffizienten in einer objektorientierten Programmiersprache oder entsprechendem Pseudocode!

Lösungsvorschlag

```java
public static int binIt(int n, int k) {
  // Das Ergebnis wird als Kommazahl deklariert, da nicht alle
  // Zwischenergebnisse ganze Zahlen sind.
  double ergebnis = 1;
  while (k > 0) {
    ergebnis = ergebnis * n / k;
```

```
      n--;
      k--;
    }
    // Vor dem Zurückgeben kann das Ergebnis nun in eine ganze Zahl
    // umgewandelt werden.
    return (int) ergebnis;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java`

(c) Geben Sie die Laufzeitkomplexität der Methoden `binRek(n, k)` und `binIt(n, k)` aus den vorhergehenden beiden Teilaufgaben in O-Notation an!

Komplette Java-Klasse

```
/**
 * <a href="https://www.studon.fau.de/file2889270_download.html">Angabe: PUE_AUD_WH.pdf</a>
 * <a href="https://www.studon.fau.de/file3081306_download.html">Lösung:
↪  PUE_AUD_WH_Lsg.pdf</a>
 */
public class Binomialkoeffizient {

  /**
   * Berechnet rekursiv den Binominalkoeffizienten „n über k". Dabei muss gelten:
   * n &#x3E;= 0, k &#x3E;= 0 und n &#x3E;= k.
   *
   * @param n Ganzzahl n
   * @param k Ganzzahl k
   *
   * @return Eine Ganzzahl.
   */
  public static int binRek(int n, int k) {
    if (k == 0 || k == n) {
      return 1;
    } else {
      return binRek(n - 1, k - 1) + binRek(n - 1, k);
    }
  }

  /**
   * Berechnet iterativ den Binominalkoeffizienten „n über k". Dabei muss gelten:
   * n &#x3E;= 0, k &#x3E;= 0 und n &#x3E;= k.
   *
   * @param n Ganzzahl n
   * @param k Ganzzahl k
   *
   * @return Eine Ganzzahl.
   */
  public static int binIt(int n, int k) {
    // Das Ergebnis wird als Kommazahl deklariert, da nicht alle
    // Zwischenergebnisse ganze Zahlen sind.
    double ergebnis = 1;
    while (k > 0) {
      ergebnis = ergebnis * n / k;
```

```
      n--;
      k--;
    }
    // Vor dem Zurückgeben kann das Ergebnis nun in eine ganze Zahl
    // umgewandelt werden.
    return (int) ergebnis;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/Binomialkoeffizient.java`

Test

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class BinomialkoeffizientTest {

  public void testeRek(int n, int k, int ergebnis) {
    assertEquals(ergebnis, Binomialkoeffizient.binIt(n, k));
  }

  public void testeIt(int n, int k, int ergebnis) {
    assertEquals(ergebnis, Binomialkoeffizient.binIt(n, k));
  }

  public void teste(int n, int k, int ergebnis) {
    testeRek(n, k, ergebnis);
    testeIt(n, k, ergebnis);
  }

  @Test
  public void teste() {
    teste(0, 0, 1);

    teste(1, 0, 1);
    teste(1, 1, 1);

    teste(2, 0, 1);
    teste(2, 1, 2);
    teste(2, 2, 1);

    teste(3, 0, 1);
    teste(3, 1, 3);
    teste(3, 2, 3);
    teste(3, 3, 1);

    teste(4, 0, 1);
    teste(4, 1, 4);
    teste(4, 2, 6);
    teste(4, 3, 4);
    teste(4, 4, 1);
  }
}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/examen/examen_46115/jahr_2014/fruehjahr/BinomialkoeffizientTest.java`