Einzelprüfung „Theoretische Informatik / Algorithmen / Datenstrukturen (nicht vertieft)“

Einzelprüfungsnummer 46115 / 2014 / Frühjahr

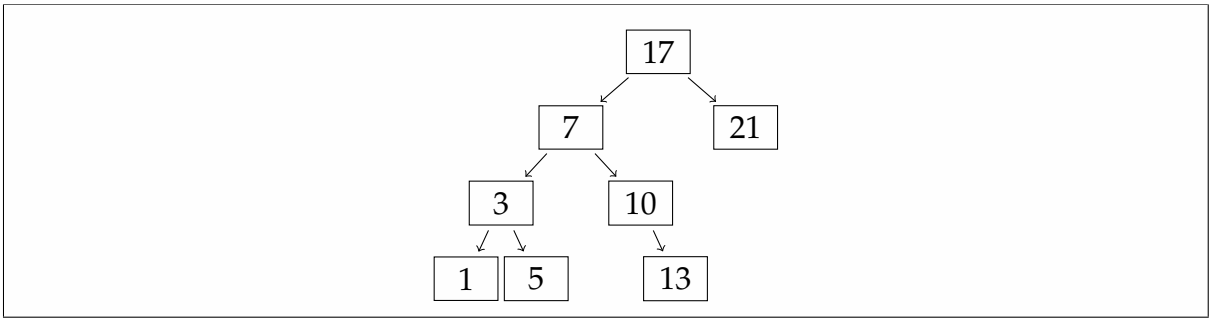
Thema 2 / Aufgabe 3

(*Binärer Suchbaum 17, 7, 21, 3, 10, 13, 1, 5*)

Stichwörter: Binärbaum

(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!

Lösungsvorschlag

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“

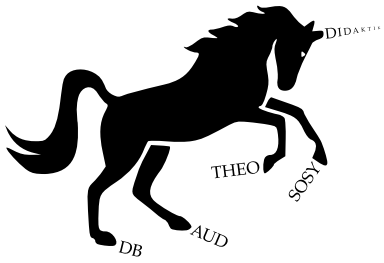
```
class Knoten {
  public int wert;
  public Knoten links;
  public Knoten rechts;
  public Knoten elternKnoten;

  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`



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 allein! 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 dieser Aufgabe kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben-tex/blob/main/Examen/46115/2014/03/Thema-2/Aufgabe-3.tex`