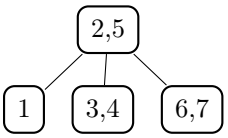
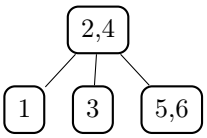
Tutoraufgabe 1 (2–3–4–Bäume):

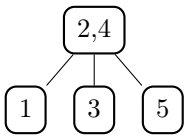
a) Löschen Sie den Wert 3 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:



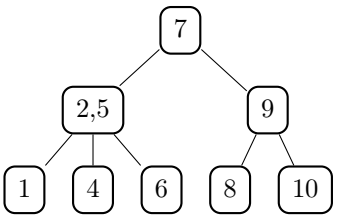
b) Löschen Sie den Wert 3 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:



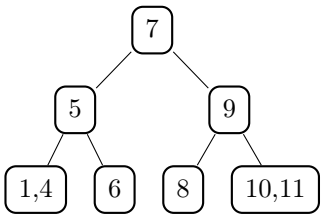
c) Löschen Sie den Wert 4 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:



d) Löschen Sie den Wert 7 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:

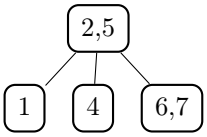


e) Löschen Sie den Wert 9 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:

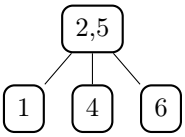


Lösung:

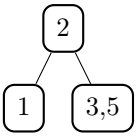
a) Schritt 1: Lösche 3



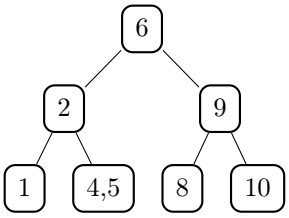
b) Schritt 1: Lösche 3



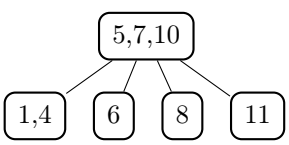
c) Schritt 1: Lösche 4



d) Schritt 1: Lösche 7

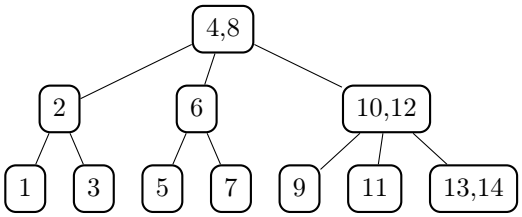


e) Schritt 1: Lösche 9

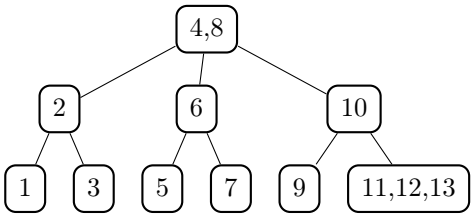


Aufgabe 2 (2–3–4–Bäume): (1 + 1 + 1 = 3 Punkte)

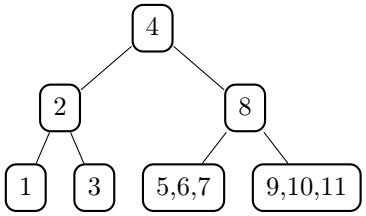
a) Löschen Sie den Wert 13 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:



b) Löschen Sie den Wert 9 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:

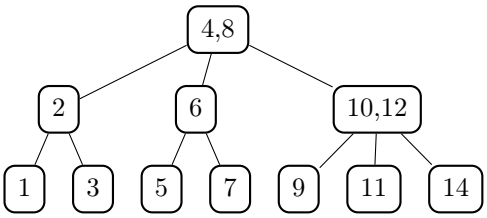


c) Löschen Sie den Wert 4 aus dem folgenden 2–3–4–Baum und geben Sie den dabei entstehenden Baum an:

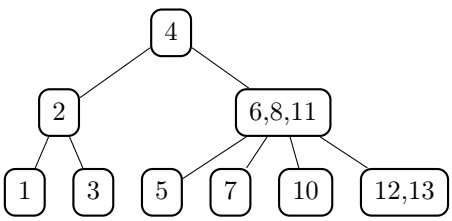


Lösung:

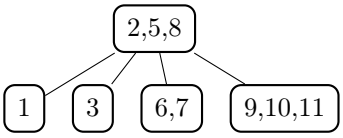
a) Schritt 1: Lösche 13



b) Schritt 1: Lösche 9



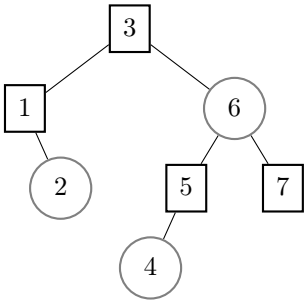
c) Schritt 1: Lösche 4



Tutoraufgabe 3 (Rot–Schwarz–Bäume):

a) Fügen Sie die Schlüssel 1, 3, 5, 6, 6, 4 und 2 in dieser Reihenfolge in einen initial leeren Rot-Schwarz Baum ein. Geben Sie den Baum direkt nach dem Einfügen sowie nach jeder Rotation an.

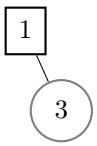
b) Löschen Sie den Knoten, den die Suche nach dem Schlüssel 6 ausgibt, aus dem folgenden Rot-Schwarz Baum. Geben Sie den Baum direkt nach dem Löschen sowie nach jeder Rotation an.



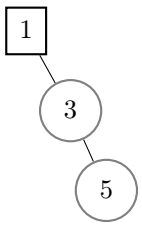
Lösung:

a) füge 1 ein

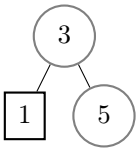
füge 3 ein



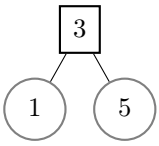
füge 5 ein



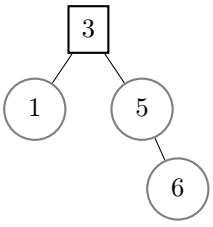
Fall 3: rotiere 1 nach links



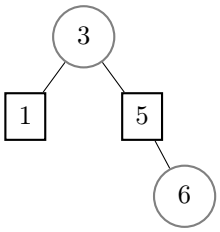
Fall 3: umfärben



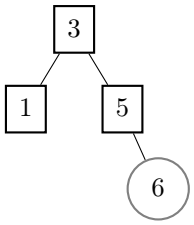
füge 6 ein



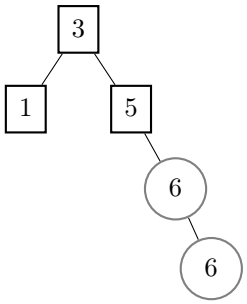
Fall 1: umfärben



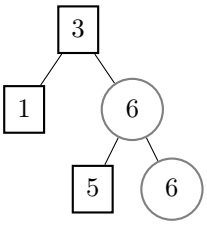
Wurzel schwarz färben



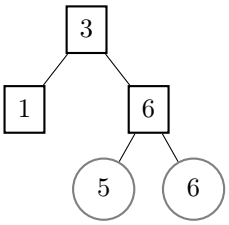
füge 6 ein



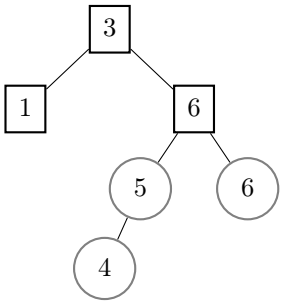
Fall 3: rotiere 5 nach links



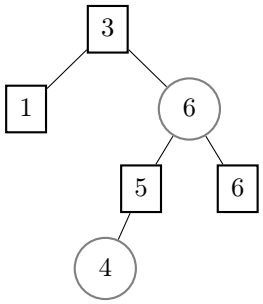
Fall 3: umfärben



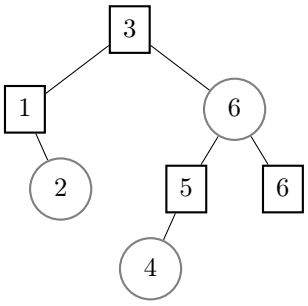
füge 4 ein



Fall 1: umfärben

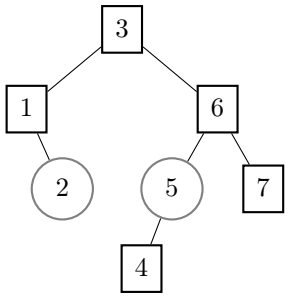


füge 2 ein

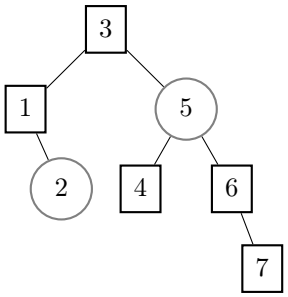


b) Wir löschen den Knoten mit dem nächst größeren Wert 7 und fügen diesen Wert dann in den zu löschenden Knoten ein.

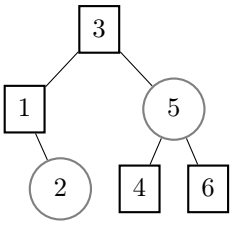
Fall 4: umfärben



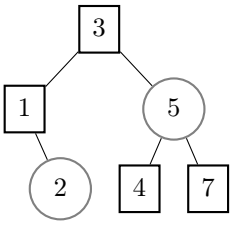
Fall 4: rotiere 6 nach rechts



ersetze 7 durch rechtes Kind

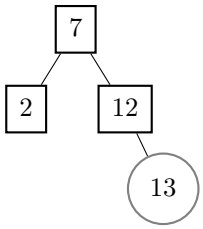


füge 7 in den zu löschenden Knoten ein

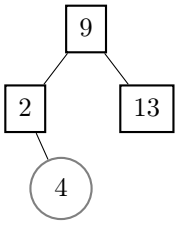


Aufgabe 4 (Rot–Schwarz–Bäume): (1,5 + 1,5 + 1,5 + 1,5 = 6 Punkte)

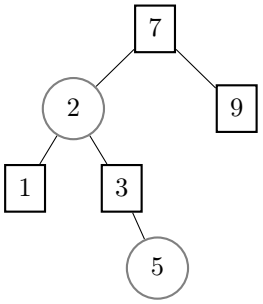
a) Fügen Sie den Schlüssel 13 in den folgenden Rot-Schwarz Baum ein. Geben Sie den Baum direkt nach dem Einfügen sowie nach jeder Rotation an.



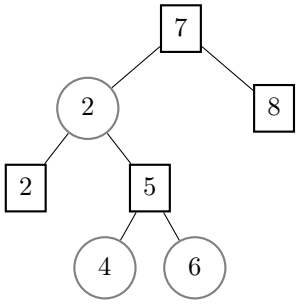
b) Fügen Sie den Schlüssel 3 in den folgenden Rot-Schwarz Baum ein. Geben Sie den Baum direkt nach dem Einfügen sowie nach jeder Rotation an.



c) Löschen Sie den Knoten, den die Suche nach dem Schlüssel 9 ausgibt, aus dem folgenden Rot-Schwarz Baum. Geben Sie den Baum direkt nach dem Löschen sowie nach jeder Rotation an.

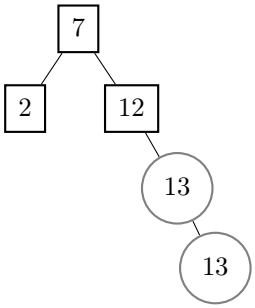


d) Löschen Sie den Knoten, den die Suche nach dem Schlüssel 2 ausgibt, aus dem folgenden Rot-Schwarz Baum. Geben Sie den Baum direkt nach dem Löschen sowie nach jeder Rotation an.

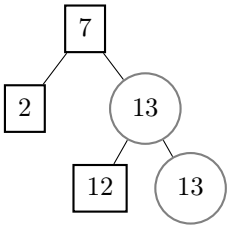


Lösung:

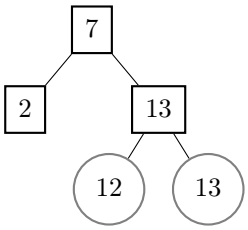
a) füge 13 ein



Fall 3: rotiere 12 nach links

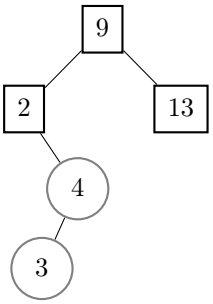


Fall 3: umfärben

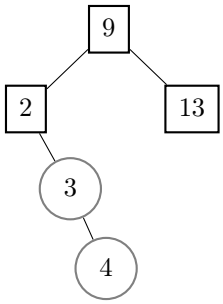


b)

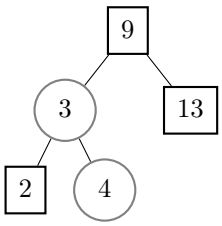
füge 3 ein



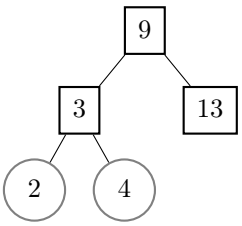
Fall 2: rotiere 4 nach rechts



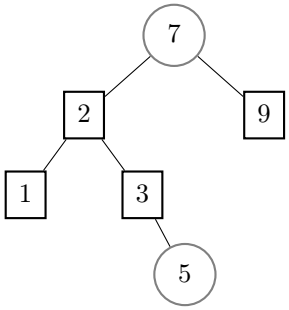
Fall 3: rotiere 2 nach links



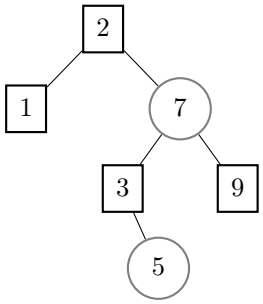
Fall 3: umfärben



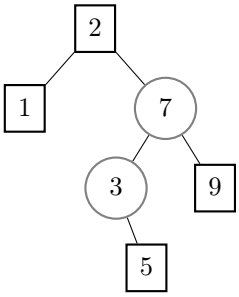
c) Fall 1: umfärben



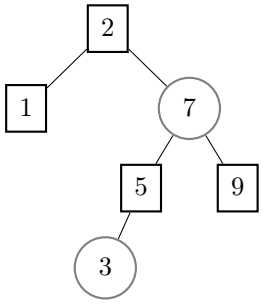
Fall 1: rotiere 7 nach rechts



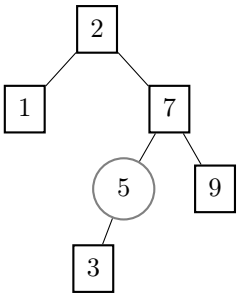
Fall 3: umfärben



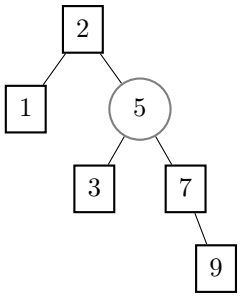
Fall 3: rotiere 3 nach links



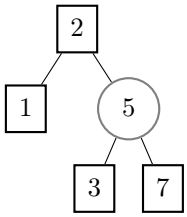
Fall 4: umfärben



Fall 4: rotiere 7 nach rechts

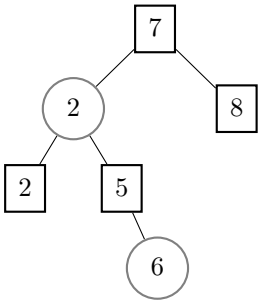


ersetze 9 durch rechtes Kind

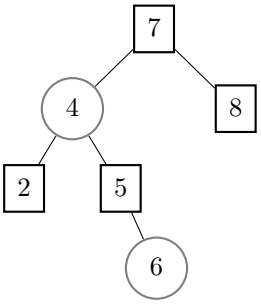


d) Wir löschen den Knoten mit dem nächst größeren Wert 4 und fügen diesen Wert dann in den zu löschenden Knoten ein.

ersetze 4 durch rechtes Kind



füge 4 in den zu löschenden Knoten ein



Aufgabe 5 (Knobeleien): (10 Punkte)

Gegeben sei die Datenstruktur `234Tree`, welche den Datentyp 2–3–4–Baum implementieren soll.
Ein Objekt vom Typ `234Tree` hat die folgenden Attribute:

- `234Tree[] children`: Das Array, welches die Kinder des Knotens enthält.
- `int[] keys`: Das Array, welches die Schlüssel des Knotens enthält.
- `int filled`: Der aktuelle Füllgrad des Knotens.
- `boolean leaf`: Gibt an, ob der Knoten ein Blatt ist.

Die Folgenden Funktionen sind definiert:

- `234Tree()`: Konstruktor, welcher einen leeren Knoten erzeugt.
- `234Tree(234Tree[] children, int[] keys)`: Konstruktor, welcher einen Knoten mit den Kindern in `children` (in genau der Reihenfolge) und den Schlüsseln `keys` (in genau der Reihenfolge) erzeugt.
  Vorbedingung: children.size() = keys.size() + 1

Zusätzlich können Sie folgende Funktionen nutzen:

- Die Funktion `void insertAt(Element[] array, Element key, int pos)` fügt einen Schlüssel `key` vom Typ `Element` an der Position `pos` in das Array `array` ein. Dabei werden nachfolgende Elemente nach hinten verschoben.
- Die Funktion `int findKeyPos(int[] array, int key, int startpos, int endpos)` findet in dem Array `array` die Einfügeposition (Rückgabewert) für den Schlüssel `key` zwischen den Array–Indizes `startpos` und `endpos`.

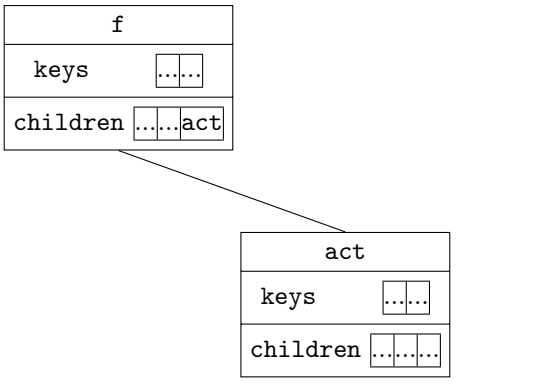
a) Implementieren Sie die Funktion `boolean split(234Tree father, int fatherIndex)`, welche **einen Split** eines Knotens wie aus der Vorlesung durchführt. Sie dürfen dabei davon ausgehen, dass der aktuelle Knoten den maximalen Füllgrad hat. Der Rückgabewert gibt dabei an, ob in der Wurzel (`true`) oder im Innern des Baumes (`false`) gesplittet wurde. Implementieren Sie die Funktion so, dass nach einem Split an der Wurzel die neue Wurzel der aktuelle Knoten ist, bei einem Split im innern des Baumes soll der aktuelle Knoten die Wurzel des neuen linken Teilbaums werden.

b) Implementieren Sie die Funktion `void insert(int key, 234Tree father, int fatherIndex)`, welche das Einfügen des Schlüssels `key` in einen 2–3–4–Baum–Knoten mit dem Vaterknoten `father` durchführen soll. `fatherIndex` ist dabei die Position des Verweises auf den aktuellen Knoten im `children`–Array des Vaterknotens.
Ein Einfügen in den gesamten 2–3–4–Baum findet also durch den Aufruf `insert(key, null, 0)` statt. Sie sollen dabei die Funktion `boolean split(234Tree father, int fatherIndex)` aus der vorherigen Aufgabe benutzen und wie in der Vorlesung sämtliche Knoten auf dem Weg zum Einfügepunkt splitten.

Hinweise:

- *Da bei einem Split innerhalb des Baumes nicht klar ist, welcher Knoten nach dem Split der aktuelle ist, nutzen Sie den Vaterknoten, um rekursiv weiter abzusteigen.*
- *Die Parameter* `father` *und* `fatherIndex` sind lediglich für die Methode `boolean split(234Tree father, int fatherIndex)` und den Fall eines Splits im innern des Baumes interessant.

Beispiel (`act` sei der aktuelle Knoten):



`father = f, fatherIndex = 2`

Lösung:

a)
```
boolean split(234Tree father, int fatherindex) {
  234Tree right = new 234Tree();
  right.keys[0] = this.keys[2];
```

```
    right.children[0] = this.children[2];
    right.children[1] = this.children[3];
    right.filled = 1;

    if (father == null) { // Split an der Wurzel
      234Tree left = new 234Tree();
      left.keys[0] = this.keys[0];
      left.children[0] = this.children[0];
      left.children[1] = this.children[1];
      left.filled = 1;

      if (this.leaf) {
        left.leaf = true;
        right.leaf = true;
        this.leaf = false;
      }
      this.keys[0] = this.keys[1];
      this.children[0] = left;
      this.children[1] = right;
      this.filled = 1;
      return true;
    } else { // Split im Innern
      if (this.leaf) {
        right.leaf = true;
      }
      this.filled = 1;
      insertAt(father.keys, this.keys[1], fatherIndex);
      insertAt(father.children, right, fatherIndex + 1 );
      father.filled++;

      return false;
    }
  }
```

b)

```
  void insert(int key, 234Tree father, int fatherIndex) {
    // Beim Absteigen Knoten splitten
    if (this.filled == 3) {
      boolean rootSplit = this.split(father, fatherIndex);
      if(rootSplit) { // an der Wurzel gesplittet
        this.insert(key, null, 0);
      }
      else { // im innern gesplittet
        int index = findKeyPos(father.keys, key, 0, father.filled - 1);
        father.children[index].insert(key, father, index);
      }
    }
    int index = findKeyPos(this.keys, key, 0, this.filled - 1);
    if (this.leaf) {
      insertAt(this.keys, key, index);
      this.filled++;
    } else {
      this.children[index].insert(key, this, index);
    }
  }
```