Algorithms and Data structures
Algorithmen und Datenstrukturen

19. Mai 2018

Praktikum 3 – B-Bäume

Version 1.0 – deutsch

Submission Deadline: 3. Juni 2018 @ 23:59

Submission System:
`https://aud.cdc.informatik.tu-darmstadt.de/cgi-bin/index.cgi`

1 Einleitung

In diesem Praktikum betrachten wir das gleiche Bibliothekensystem aus Praktikum 1 (QuickSort) welches RFID-Tags nutzt. In dem vorliegenden Praktikum werden gelesenen Werte nicht in einer zentralen Datei gespeichert, sondern in einer B-Baum-Datenstruktur gespeichert um die Verwaltung der Daten zu vereinfachen. Für die B-Baum-Datenstruktur sind zum Beispiel folgende grundlegende Operationen implementiert: Einfügen, Suchen und Löschen von Einträgen. Das Format der Einträge welche in den B-Baum eingefügt werden sollen, ist das gleiche wie im Praktikum 1 (QuickSort):

```
Book_serial_number;ReaderID;STATUS.
```

Die `Book_serial_number` und die `ReaderID` bilden zusammen den *key* der Einträge in der B-Baum-Datenstruktur (der *key* ensteht durch die Verkettung der String-Werte). Der `STATUS` repräsentiert die *Daten*, die in jedem Eintrag gespeichert sind.

2 Task

Ihre Aufgabe ist es eine Java-Klasse (`B_Tree`) zu entwickeln welche die in der Vorlesung vorgestellte B-Baum-Datenstruktur implementiert, um die von den RFID-Lesegeräten gelesenen Informationen zu speichern. Ein B-Baum ist ein vollständig ausbalancierter Baum. Das bedeutet, dass jeder Weg von der Wurzel bis zum Blatt die gleiche Länge hat. Jeder Knoten in einem B-Baum mit minimalem Grad t (außer der Wurzel) hat mindestens t und höchstens $2t$ Kinder und damit mindestens $t-1$ und höchstens $2t-1$ Schlüssel. Die Wurzel des B-Baumes ist entweder ein Blatt oder hat mindestens zwei Kinder. Im B-Tree werden die Schlüssel und Daten sowohl in den internen Knoten als auch in den Blättern gespeichert. Die in den Knoten des B-Trees gespeicherten Schlüssel sind eindeutig. Der minimale Grad des B-Trees kann nach der Initialisierung nicht mehr verändert werden. Das bedeutet, dass t nach der Initialisierung des B-Trees nicht mehr geändert werden kann.

3 The Code

Wir stellen die folgenden Java Klassen und Tests in dem Zip-Archiv **Lab3.zip** zu Verfügung, welches das Java-Projekt und seine Bausteine enthält. Die Klassen sind aufgeteilt in zwei “packages” – das “frame” und das “lab” package. Bitte implementieren Sie Ihre Lösung basierend auf dem im Zip-Archiv zur Verfügung gestellten Codegerüst.

3.1 Das ‘frame’ package

Das Abgabesystem nutzt eigene Kopien dieser Klassen wenn ihr Quellcode auf dem Server getestet wird. Mögliche lokalen Änderung die Sie an diesem package vornehmen, können vom Abgabesystem also nicht berücksichtigt werden.

3.1.1 `AllTests.java`

Dies ist die JUnit Test-Klasse. Die Testfälle werden benutzt um die Korrektheit Ihrer Lösung zu überprüfen.

3.1.2 `TestNode.java`

Diese Datei enthält die Klasse, die während des Testens genutzt wird um die Knoten zu repräsentieren.

3.1.3 `DotFileConstants.java`

Diese Datei enthält Bestandteile der `dot`-Sprache. Die `dot`-Sprache wurde bereits im Praktikum 2 (Hash-Tables) eingeführt und wird in Abschnitt 4 noch einmal erläutert.

3.1.4 `Entry.java`

Die Einträge des B-Baums werden in Objekten vom Typ `Entry` abgelegt. Die Klasse `Entry` implementiert die folgenden Schnittstellen (Änderungen an dieser Klasse sind nicht erlaubt!).

- `EntryInterface`

  Diese Schnittstelle definiert folgende Methoden:

  - `void setKey(String newKey)`: Diese Methode setzt den Wert des `keys` des Eintrags auf den übergebenen Wert `newKey`.
  - `void setData(String newData)`: Diese Methode setzt den Wert der Daten des Eintrags auf den übergebenen Wert `newData`.
  - `String getKey()`: Diese Methode gibt den Wert des `keys` des Eintrags zurück.
  - `String getData()`: Diese Methode gibt den Wert der Daten des Eintrags zurück.
  - `String toString()`: Die Methode gibt die String-Darstellung des Eintrags zurück `Book_serial_number;ReaderID;STATUS`, also zum Beispiel `Z8IG4;LDXS;OK`.

- `Comparable<Entry>` Diese Schnittstelle ist nötig damit zwei Objekte des Typs `Entry` direkt miteinander verglichen werden können.

3.1.5 `EntryInterface.java`

Diese Datei definiert die Schnittstelle `EntryInterface`.

3.2 Das 'lab' package

Dieses package enthält die Dateien, welche Sie modifizieren dürfen. Sie dürfen neue Klassen (in neuen Dateien innerhalb des packages, aber bitte nicht in subpackages), als auch neue Methoden oder Variablen hinzufügen, solange **Sie die Signatur (*Name, Parameter Typ und Anzahl*) der gegebenen Methoden nicht ändern**, da diese von den JUnit Tests verwendet werden. Änderungen außerhalb des 'lab' package werden ignoriert, wenn Sie ihren Quellcode über das Abgabesystem hochladen.

3.2.1 `B_Tree.java`

Für die `B_Tree` Klasse müssen folgende Methoden implementiert werden:

- `public B_Tree(int t)`

  Der Konstruktor der Klasse bekommt als Parameter einen Integer t, der den minimalen Grad der B-Baum-Datenstruktur darstellt. Der Wert von t kann nach dem Anlegen eines `B_Tree`-Objekts nicht mehr geändert werden.

- `public int constructB_TreeFromFile (String filename)`

  Diese Methode bekommt als Eingabeparameter den Namen einer Datei, welche eine Liste von Einträgen enthält. Die Einträge sollen in der gleichen Reihenfolge in der sie in der Liste erscheinen, in den B-Baum eingefügt werden sollen. Sie können davon ausgehen, dass sich die Datei im gleichen Verzeichnis wie das ausführbare Programm befindet. Die Datei ähnelt der Datei aus dem Praktikum 1 (QuickSort). Der Rückgabewert der Methode ist die Anzahl der *erfolgreich* in den B-Baum eingefügten Einträge. Hinweis: Die Einträge des B-Baums werden in Objekten vom Typ `Entry` gespeichert (vgl. Abschnitt 3.1.4).

  Eine Beispieldatei sieht wie folgt aus (vgl. Praktikum 1):

```
Z8IG4;LDXS;OK
0X6F9;ERSY;OK
YSI7Q;ERSY;OK
6C8IV;ERSY;Error
YSI7Q;4OO9;OK
EMBXP;GQ9Y;OK
5MXGT;7L8Q;Error
FOC9U;7L8Q;OK
XOH3X;GQ9Y;Error
XOH3X;ERSY;Error
XDYF6;P8OS;OK
GFN81;7L8Q;Error
FOC9U;7L8Q;OK
WN178;GQ9Y;OK
```

Um diese Einträge in die B-Tree-Struktur einzufügen, verwenden Sie bitte die folgende Methode:

- `public boolean insert(Entry insertEntry)`

  Diese Methode fügt den Eintrag `insertEntry` an der richtigen Stelle in den B-Baum ein. Beachten Sie, dass es bei dieser Methode zu Überläufen kommen kann, beispielsweise wenn Sie einen Eintrag in ein Blatt einfügen wollen, das bereits $2t-1$ Einträge enthält. Diese Methode gibt `true` zurück, wenn das Einfügen des Eintrags `insertEntry` erfolgreich war. Sie gibt `false` zurück, wenn der *key* dieses Eintrags bereits im B-Baum existiert.

- `public Entry delete(String deleteKey)`

  Diese Methode löscht den Eintrag mit dem `key deleteKey` aus der B-Baum-Datenstruktur. Unterscheiden Sie bei dieser Methode bite die folgenden beiden Fälle:

  1. Der Eintrag mit dem `key deleteKey` befindet sich in einem Blatt.
  2. Der Eintrag mit dem `key deleteKey` befindet sich in einem Knoten.

  Diese Methode gibt den Eintrag mit `deleteKey` als key zurück, wenn das Löschen erfolgreich war und `null`, wenn der `key deleteKey` in keinem Eintrag des B-Baums gefunden wurde.

  Bitte beachten Sie, dass der B-Baum nach dem Löschen eines Elements nicht immer eindeutig ist. Daher bitten wir Sie die nachfolgenden Anweisungen umzusetzen:

  1. Versuchen Sie immer zuerst “Rotation” statt “Verschmelzung”. Falls die Rotation nicht möglich ist, wird eine Verschmelzung durchgeführt.
  2. Versuchen Sie immer zuerst mit dem linken Nachbarn zu rotieren.
  3. Versuchen Sie immer zuerst mit dem linken Nachbarn zu verschmelzen.
  4. Wenn Sie einen Eintrag aus einem internen Knoten löschen wollen, ersetzen Sie diesen durch den Eintrag mit dem nächst größeren Schlüssel als den gelöschten.

- `public Entry find(String searchKey)`

  Diese Methode sucht in dem B-Baum nach dem Eintrag mit dem `key searchKey`. Die Methode gibt den Eintrag mit dem key `searchKey` zurück, wenn ein solcher Eintrag gefunden wird. Ansonsten gibt sie `null` zurück.

- `public ArrayList<String> getB_Tree()`

  Diese Methode gibt eine `ArrayList<String>` zurück, die den B-Baum enthält. Die Ausgabe soll direkt interpretierbar sein, wie in Abschnitt 4 beschrieben. Jedes Element in der `ArrayList` entspricht einer Zeile des Ausgabebaums. Für den B-Baum in Abbildung 4 würde die Ausgabe-ArrayList die Länge 12 haben, wie in Abbildung 3 anhand der Nummerierung zu erkennen ist. Die Knoten des B-Baums sollten nur die Schlüssel der Einträge und nicht die Daten enthalten.

- `public int getB_TreeHeight()`

  Diese Methode gibt die Höhe des B-Baumes zurück. Wenn der B-Tree leer ist oder nur den Wurzelknoten enthält, soll diese Methode 0 zurückgeben.

- `public ArrayList<Entry> getInorderTraversal()`

  Diese Methode traversiert den B-Baum der Reihe nach und fügt jeden Eintrag zu einer `ArrayList<Entry>` hinzu, so dass die zurückgegebene ArrayList die Einträge des B-Baums in aufsteigender Reihenfolge enthält.

- `public int getB_TreeSize()`

  Diese Methode liefert die Anzahl der Einträge im B-Baum –nicht die Anzahl der Knoten– zurück. Für den B-Baum in Abbildung 4 sollte diese Methode beispielsweise 13 zurückgeben.

3.2.2 `B_TreeNode.java`

Diese Klasse kann verwendet werden, um Knoten des B-Baums zu implementieren, die die Einträge speichern. Es steht Ihnen frei diese Klasse zu verwenden oder nicht.

3.3 Test Files

Wir stellen Ihnen die folgende Testdatei zur Verfügung:

- `TestFile1.txt`

Wir empfehlen Ihnen, Ihre Lösung auch mit zusätzlichen Eingabedateien zu testen. Beachten Sie, dass Sie Ihre eigenen JUnit-Testfälle schreiben müssen, um mit Ihren benutzerdefinierten Eingabegraphen arbeiten zu können. Denken Sie über die Annahmen nach, die bei der Eingabe gemacht wurden. Um sicherzustellen, dass Ihre Lösung funktioniert, testen Sie sie mit allen bereitgestellten Testfällen. Neben den angegebenen Eingabedateien testet das Abgabesystem Ihre Lösung mit mehreren zusätzlichen Eingabedateien, um die Korrektheit Ihres Programms zu bestätigen.

4 Output format: dot

Als Ausgabeformat für dieses Praktikum verwenden wir eine Untermenge der im Praktikum 2 (Hash-Tabellen) eingeführten `dot`-Sprache. Diese Untermenge wird im Folgenden beschrieben. Wir stellen Ihnen ein Beispiel zur Verfügung, um den Teil der `dot`-Sprache zu verstehen, den Sie für dieses Praktikum benötigen. Der Einfachheit halber verwenden wir im Beispiel ganze Zahlen als Schlüssel (anstelle von Strings).

4.1 dot language

Die Darstellung eines Knotens des B-Baum in Abbildung 2 in `dot`-Sprache ist in Abbildung 1 dargestellt. Die mit "*" gekennzeichneten Felder in der Knotendarstellung sind die Zeiger auf die Knoten der nächsten Ebene in der B-Baum-Datenstruktur. $f0$, $f1$, $f2$, ... sind die Namen der Felder in der Knotendarstellung.

```
digraph{
node[shape=record];
node1[label="<f0>*|<f1>42|<f2>*|<f3>45|<f4>*|<f5>77|<f6>*"];
}
```

Abbildung 1: Code für Abbildung 2

Abbildung 2: Beispiel eines B-Baum Knotens

4.2 Example

Der Graph in Abbildung 4 stellt ein Beispiel für einen B-Baum mit minimalem Grad 3 dar, wobei die `keys` der Einträge Integer sind. Der B-Baum entsteht durch Einfügen der folgenden ganzen Zahlen in der angegebenen Reihenfolge: 11, 18, 43, 47, 42, 31, 55, 62, 71, 77, 83, 91, 99. Abbildung 3 stellt den entsprechenden Code in `dot`-Sprache dar. Alle Ausgabebäume sollten dem Format in Abbildung 3 folgen. Das bedeutet, dass die Blätter aussehen wie interne Knoten, aber keine Zeiger auf andere Knoten besitzen. Die Reihenfolge der Einträge in der Punktdarstellung ist mit Ausnahme der ersten beiden und der letzten nicht relevant. Das Format der Knoten des Ausgabebaums sollte nur die `keys` und keine Daten enthalten. Der Wurzelknoten eines B-Baums in diesem Praktikum sollte wie in Abbildung 3 explizit als *root* bezeichnet werden. Um einen Zeiger von einem Knoten zu einem anderen in der nächsten Ebene zu zeichnen, müssen Sie den Namen des Feldes mit dem Zeiger wie in Abbildung 3 angeben (beispielsweise `root:f0->node2;`).

```
1.  digraph{
2.  node[shape=record];
3.  root[label="<f0>*|<f1>42|<f2>*|<f3>55|<f4>*|<f5>77|<f6>*"];
4.  node2[label="<f0>*|<f1>11|<f2>*|<f3>18|<f4>*|<f5>31|<f6>*"];
5.  node3[label="<f0>*|<f1>43|<f2>*|<f3>47|<f4>*"];
6.  node4[label="<f0>*|<f1>62|<f2>*|<f3>71|<f4>*"];
7.  node5[label="<f0>*|<f1>83|<f2>*|<f3>91|<f4>*|<f5>99|<f6>*"];
8.  root:f0->node2;
9.  root:f2->node3;
10. root:f4->node4;
11. root:f6->node5;
12. }
```

Abbildung 3: Beispielcode eines B-Baums in `dot`-Sprache

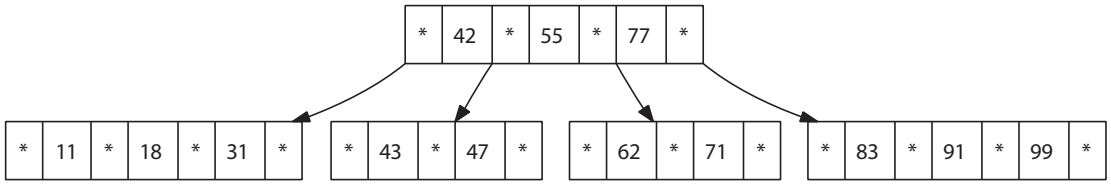

Abbildung 4: Ein einfacher B-Baum mit $t = 3$ entsprechend dem Code in Abbildung 3