Übungen zu Algorithmen und Datenstrukturen für Medizintechnik
Sommersemester 2021
René Raab, Robert Richer

Lehrstuhl für Maschinelles Lernen und Datenanalytik
Department Artificial Intelligence in Biomedical Engineering (AIBE)
(https://www.mad.tf.fau.de)

Aufgabenblatt 9 vom 27. Juni 2021, Abgabe am 11. Juli 2021, 22:00 Uhr

---

Aufgabe 9.1: Bäume – Theorie

Punkte siehe StudOn
AVL-Bäume, Halden

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 9.2: Sortieren – Theorie

Punkte siehe StudOn
Sortierverfahren

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 9.3: RadixSort

10 Punkte
Sortieralgorithmen

In dieser Aufgabe werden Sie das aus der Vorlesung bekannte RadixSort-Sortierverfahren implementieren. Der RadixSort ist eine Weiterentwicklung des BucketSort-Sortierverfahrens. Der große Vorteil des RadixSort ist die geringere Speicherauslastung, da nicht jeder mögliche Schlüsselwert einen eigenen Behälter bekommt. Um die Aufgabe nicht unnötig kompliziert zu machen, sollen hier nur Strings sortiert werden können.

1. Erstellen Sie ein neues Projekt `09-RadixSort` mit einer Klasse `RadixSort`.
2. Implementieren Sie nun eine Methode `void sort(List<String>, int)`. Diese Methode bekommt eine Liste mit generischen, zu sortierenden Elementen (`words`), sowie die Anzahl der zu sortierenden Stellen (`digits`) übergeben. Die Methode soll `words` in-place sortieren, d.h. die übergebene Liste soll nach Aufruf der Methode sortiert sein ohne, dass die Methode etwas zurückgeben muss.

   Hinweis: Der Einfachheit halber können Sie davon ausgehen, dass alle zu sortierenden Elemente die gleiche Anzahl an `digits` haben.
3. Um die Listenelemente sortieren zu können, benötigen Sie eine Datenstruktur, welche die einzelnen Sortierbehälter speichert. Wählen Sie für diese Datenstruktur ein »Array aus `ArrayList`«, sodass die Anzahl der Behälter fest ist, jeder Behälter allerdings eine beliebige Anzahl an Elementen speichern kann.

Hinweis: Ein Array aus String-`ArrayLists` können Sie wie folgt erzeugen:
Die Variablen `amount` und `buckets` müssen Sie natürlich an Ihren Code anpassen.

```
@SuppressWarnings("unchecked")
List<String>[] buckets = (List<String>[]) new ArrayList[amount];
```

4. Die Anzahl der benötigten Behälter hängt natürlich davon ab, welche Elemente Sie sortieren möchten. Da wir uns in dieser Aufgabe auf Strings beschränken (und auch nur Eingaben mit Zeichen aus dem ASCII-Zeichensatz erlauben) beschränken wir die Anzahl der Behälter auf 256 (ein Behälter für jedes Zeichen in der erweiterten ASCII-Tabelle).

5. Führen Sie nun mithilfe der übergebenen Liste und den Behältern das in den Vorlesungsfolien beschriebene Sortierverfahren durch.

6. Erstellen Sie dafür eine Hilfsmethode `private int getBucket(String value, int position)`, welche das zu sortierende Element und die Stelle, nach der sortiert werden soll, übergeben bekommt und berechnet, in welchen Behälter das Elmement eingefügt werden muss.

Wichtig: Achten Sie darauf, dass Ihre RadixSort-Implementierung wie die in der Vorlesung beschriebene RadixSort-Variante nach dem *Least Significant Digit (LSD)*-Verfahren funktioniert, die Elemente also von *rechts nach links* sortiert werden!

7. Testen Sie Ihr Programm ausführlich! Verwenden Sie als Tests die zwei folgenden Anwendungen:

   - **IDM-Kennungen der FAU:**
     Wie Ihnen vielleicht bereits aufgefallen ist, bestehen IDM-Kennungen der FAU, mit der Sie sich in Portale wie StudOn oder meinCampus einloggen, standardmäßig aus einer 8-stelligen Kombination aus Buchstaben und Zahlen (z. B. ab12wxyz). Legen Sie eine Liste mit mehreren IDM-Kennungen an und lassen Sie diese im Anschluss sortieren. Geben Sie die Liste vor und nach dem Sortiervorgang auf `stdout` aus.

   - **Deutsche Postleitzahlen:**
     Legen Sie eine Liste mit mehreren 5-stelligen Postleitzahlen an und lassen Sie diese im Anschluss sortieren. Geben Sie die Liste vor und nach dem Sortiervorgang auf `stdout` aus.

8. Geben Sie die Datei `RadixSort.java` im EST ab.

Aufgabe 9.4: Binäre Suchbäume

20 Punkte

Suchbaumoperationen, Traversierung

Auf dem vorletzten Übungsblatt haben Sie eine sortierte Liste als doppelt verkettete Liste realisiert. In dieser Aufgabe möchten wir nun eine sortiere Menge implementieren – jedes Element darf also nur einmal vorkommen. Eine spezialisierte und effiziente Datenstruktur für diese Aufgabe ist der binäre Suchbaum. Er ermöglicht eine im Durchschnitt (“average case”) logarithmische Komplexität für typische Operationen wie Einfügen und Entfernen. Wie er funktioniert, wurde in der Tafelübung behandelt.

In dieser Aufgabe sollen Sie das vorgegebene `BSTInterface` anhand seiner Dokumentation in einer eigenen Klasse `BinarySearchTree` implementieren. Der binäre Suchbaum soll nach `Integer`-Werten suchen können.

Achtung: Verwenden Sie intern keine Methoden aus der Java-API außer die der folgenden Klassen und Interfaces: `List<E>`, `ArrayList<E>`, `NoSuchElementException` und `String`. Zum Testen dürfen Sie zusätzlich `Random` benutzen.

1. Legen Sie ein neues Projekt `09-BinarySearchTree` an.

2. Die innere Klasse `TreeNode` (Baumknoten) soll einen Konstruktor mit einem Parameter besitzen, der einen übergebenen `int`-Wert im Attribut `int value` speichern soll. Zudem soll ein Knoten Referenzen auf seine beiden Kinder (`left`, `right`) sowie seinen Elternknoten (`parent`) speichern.

3. Die Methoden `insert()` und `exists()` **müssen *rekursiv*** implementiert werden. Die iterativen Varianten kennen Sie ja bereits aus den Übungsfolien ☺.

4. Falls das einzufügende Element bereits im Baum vorhanden ist, soll eine `ElementExistsException` geworfen werden. Dafür müssen Sie Ihre eigene `Exception`-Klasse schreiben, die von `java.lang.Exception` erbt. Überlegen Sie sich eine sinnvolle Fehlernachricht!

5. Wie Sie `remove(int value)` umsetzen, ist Ihnen überlassen. Sie erleichtern sich aber die Arbeit sehr, wenn Sie Ihren Code nach den drei möglichen Fällen gliedern und sich einige Hilfsmethoden dazu anlegen:

   Hinweis: Eventuell könnte ein Blick in die Vorlesungsfolien hilfreich sein ☺

   - Bestimmung der *Anzahl der Kinder* eines Knotens, zur Unterscheidung der Fälle (⇝ `int numChildren(TreeNode node)`).
   - *Finden* des Knotens mit dem gesuchten Wert, falls vorhanden (⇝ `TreeNode find(TreeNode node, int value)`).
   - *Entfernen* eines **Knotens**. Es bietet sich an, hier den Großteil der Logik zu implementieren. (⇝ `void remove(TreeNode node)`).
   - *Ersetzen* der Referenz auf einen Knoten *im Elternknoten* (Spezialfall: Wurzel). (⇝ `void replaceInParent(TreeNode node, TreeNode newNode)`).

   Achtung: Vergessen Sie nicht, Referenzen auf Elternknoten konsistent zu halten!

6. Legen Sie zum Schluss eine `main`-Methode an, um Ihre Implementierungen zu testen. Testen und vergleichen Sie, was passiert, wenn Sie den Baum in inorder, preorder und postorder Reihenfolge durchlaufen (⇝ Methoden `inOrderList`, `preOrderList` und `postOrderList`!).

   Hinweis: Zur grafischen Darstellung des Baumes steht Ihnen zu dieser Aufgabe auch eine Java-Bibliothek `TreeViever` zur Verfügung. Wenn Sie die Grundlagen des Baumes implementiert haben, können Sie die Datei `tree-viewer.jar` in Ihr Projekt einbinden und anschließend die Methode `TreeViewer.displayTree(BinarySearchTree tree)` aufrufen.

   Entfernen Sie vor Abgabe alle Verwendungen des `TreeViewer` (auch den Import!) oder kommentieren Sie diese aus. Das EST kann Ihre Abgabe andernfalls nicht verarbeiten und Sie erhalten 0 Punkte.

7. Geben Sie die Klassen `BinarySearchTree.java` und `ElementExistsException.java` ab.

Sollte Ihr Programm nicht übersetz- bzw. ausführbar sein, wird die Lösung mit 0 Punkten bewertet. Stellen Sie also sicher, dass IntelliJ IDEA keine Fehler in Ihrem Programm anzeigt, Ihr Programm übersetz- und ausführbar ist sowie die in der Aufgabenstellung vorgegebenen Namen und Schnittstellen *exakt* eingehalten werden. Geben Sie am Schluss die Dateien `RadixSort.java` und `BinarySearchTree.java, ElementExistsException.java` über die EST-Webseite ab. Wenn Sie die Aufgabe zusammen mit einem Übungspartner bearbeitet haben, geben Sie im EST unbedingt dessen Gruppenabgabe-Code an! Kontrollieren Sie, ob Ihre Namen am Anfang aller Dateien angegeben sind – schreiben Sie im Quellcode Ihre Angaben in einen Kommentar. Im EST-Abgabesystem können Sie modifizierte Dateien mehrfach abgeben. Nur die zuletzt hochgeladene Version wird bewertet.