FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

10. Übung

Abgabe bis 23.01.2017, 10:00 Uhr

Einzelaufgabe 10.1: Allgemeine Bäume und Binärbäume — 6 EP

Gegeben sind die folgenden Werte: 4, 8, 3, 6, 5, 9, 2, 7.

a) Erstellen Sie aus obigen Werten einen Baum mit maximaler Höhe. Geben Sie die Höhe h des Baumes an.

b) Erstellen Sie aus obigen Werten einen Baum mit minimaler Höhe. Geben Sie die Höhe h des Baumes an.

c) Erstellen Sie aus obigen Werten einen Binärbaum minimaler Höhe.

d) Welche minimale und maximale Höhe ergibt sich für einen allgemeinen Baum mit n Werten?

e) Welche minimale und maximale Höhe ergibt sich für einen binären Baum mit n Werten?

Bitte geben Sie ihre Lösung als `tree.pdf` ab.

Einzelaufgabe 10.2: Binäre Suchbäume — 8 EP

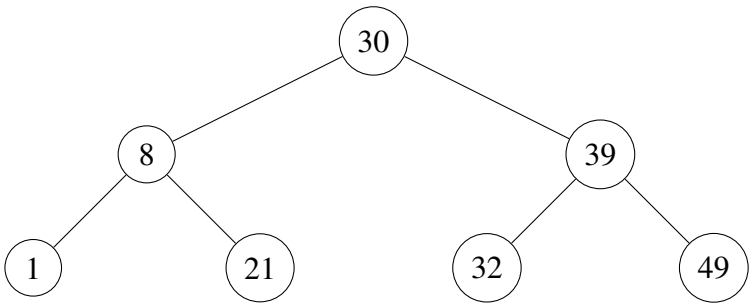
Gegeben sind die folgenden Werte: 41, 44, 24, 27, 7, 34, 9, 11, 35, 19

a) Fügen Sie oben genannte Werte der Reihe nach in einen binären Suchbaum ein. Geben Sie das Bild des Endzustandes ab.

b) Bei welcher Eingabereihenfolge der oben gegebenen Werte wird die Höhe des binären Suchbaums maximal?

c) Geben Sie die mittlere Anzahl der benötigten Zugriffe (Mittelwert) für die Suche in den zwei Bäumen aus a) und b) an. Es wird angenommen, dass die gesuchten Elemente auch alle enthalten sind.

d) Gegeben ist der folgende binäre Suchbaum. Geben Sie die lineare Ordnung nach `inorder` und `preorder` Form an.



e) Löschen Sie aus dem binären Suchbaum aus der letzten Teilaufgabe den Knoten mit dem Wert 30 nach beiden in der Vorlesung genannten Varianten. Geben Sie die beiden resultierenden binären Suchbäume an.

Bitte geben Sie ihre Lösung als `sbintree.pdf` ab.

Einzelaufgabe 10.3: Streutabellen - Implementierung **23 EP**

In dieser Aufgabe werden Sie eine einfache Streutabelle mit verketteten Listen zur Kollisionsverwaltung implementieren.
Laden Sie HashMapInterface, Pair und ResizingHashMap von der AuD-Website und machen Sie sich damit vertraut.
Ihre Aufgabe ist es, die Schnittstelle `HashMapInterface<K, V>` in der Klasse `ResizingHashMap<K, V>` zu implementieren. Die generische Signatur wird im folgenden aus Gründen der Übersichtlichkeit weggelassen.
Die Nutzung von Klassen aus dem Package `java.util` sowie der Klasse `java.lang.Math` ist verboten.

a) Dem Konstruktor der Klasse `ResizingHashMap` wird eine Größenangabe übergeben. Diese bestimmt die Größe der internen Tabelle `buckets` und ist zunächst konstant (bis zum Aufruf der Methode `resize()`). Initialisieren Sie die Streutabelle entsprechend.
Hinweis: Die Streutabelle kann durch die Verwendung von verketteten Listen zur Auflösung von Kollisionen innerhalb eines Feldes von `buckets` beliebig viele Elemente aufnehmen.

b) Die Methode `size()` gibt die Anzahl der Elemente zurück, die momentan in der Streutabelle gespeichert sind. Der Aufwand der Methode `size()` muss $\mathcal{O}(1)$ betragen. Verwenden Sie das Feld `size` für die Buchhaltung der Anzahl der Elemente.

c) Die Methode `insertEntry()` fügt den gegebenen Schlüssel-Wert-Eintrag an dem gegebenen Index in die Streutabelle ein (ohne zu überprüfen ob bereits ein Eintrag mit dem selben Schlüssel existiert). Falls das entsprechende Bucket leer ist kann der Eintrag direkt gespeichert werden. Andernfalls kommt es zu einer Kollision und der neue Eintrag muss **am Ende** der verketteten Liste angehängt werden (Verzeigerung mittels des Feldes `Pair.nextPairInBucket`). Aktualisieren Sie die interne Buchführung.

Die Verwaltung der Elemente wird in Abbildung 1 verdeutlicht. Jeder Eintrag im Array `buckets` entspricht einem Bucket und zeigt auf den ersten Eintrag (Schlüssel-Wert-Paar) einer verketteten Liste, an die im Falle einer Kollision weitere Einträge angehängt werden. Die Verzeigerung der Listenelemente in einem Bucket erfolgt über das Feld `Pair.nextPairInBucket`.

d) Die Methode `getBucketIndex()` berechnet zu einem Schlüssel-Objekt den entsprechenden Tabellenindex. Benutzen Sie dazu die `hashCode()`-Methode. Sorgen Sie dafür, dass der Hashcode auf den Indexbereich der Tabelle abgebildet wird, die im Feld `buckets` gespeichert ist. Bedenken Sie, dass `hashCode()` auch negative Werte zurückgeben kann. Bringen Sie diese Werte durch Verwendung des Modulo-Operators sowie Additionen in den richtigen Wertebereich (ohne die Verwendung von Schleifen, außerdem ist die Benutzung der Klasse `java.lang.Math` **untersagt**).

e) Implementieren Sie nun die Methode `getEntry()`. Diese soll den Eintrag mit dem Schlüssel-Wert-Paar zurückgeben, das über den übergebenen Schlüssel identifiziert wird. Der Vergleich von Schlüsselobjekten hat mit der `equals()`-Methode zu erfolgen. Ist in der Tabelle kein Paar mit passendem Schlüssel gespeichert, dann geben Sie `null` zurück.

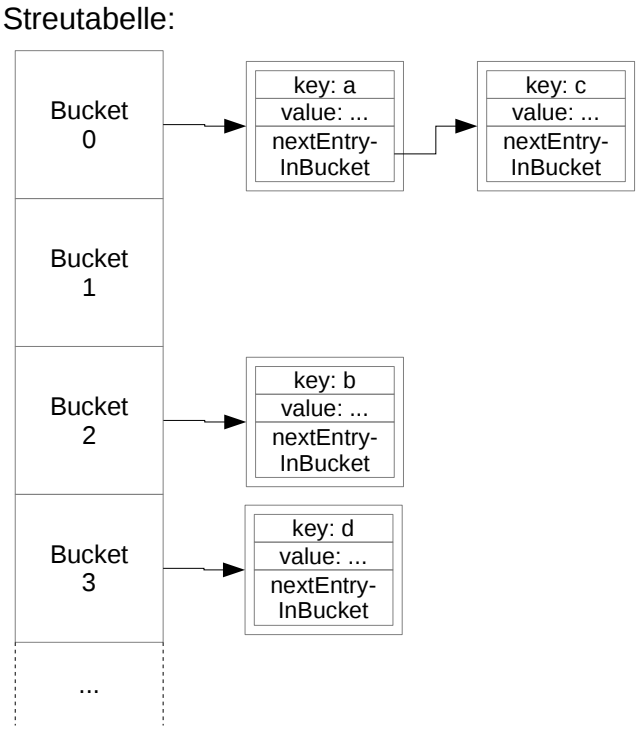


Abbildung 1: Zusammenspiel der Datenstrukturen der Streutabelle. Es wurden Elemente mit den Schlüsseln *a*, *b*, *c*, *d* eingefügt (mit einer Kollision).

f) Die Methode `get()` funktioniert im Prinzip wie die Methode `getEntry()`. Sie gibt jedoch nur den mit einem bestimmten Schlüssel assoziierten Wert zurück und nicht den ganzen Eintrag (das Paar).

g) Mit der `contains()` Methode soll überprüft werden, ob zu einem bestimmten Schlüssel ein Wert in der Tabelle gespeichert ist.

h) Implementieren Sie nun die Funktionalität für die *put*-Operation. Falls der übergebene Schlüssel bereits mit einem Wert assoziiert ist, dann wird der gespeicherte Wert aktualisiert und kein neuer Eintrag angelegt. In diesem Fall gibt die Methode `false` zurück.

Ist der übergebene Schlüssel in der Streutabelle noch nicht vertreten, dann muss zunächst ein Eintrag vom Typ `Pair` erzeugt werden und anschließend eingefügt werden (verwenden Sie hierzu die bereits von Ihnen implementierte Methode `insertEntry()`). Die Methode gibt dann `true` zurück.

i) Implementieren Sie nun die Funktionalität für die *remove*-Operation.

Suchen Sie in der Streutabelle nach dem Eintrag, der auf den übergebenen Schlüssel passt. Folgen Sie, wenn nötig, der Verzeigerung der Einträge innerhalb eines Buckets. Sollte sich kein passender Eintrag finden, geben Sie `false` zurück. Entfernen Sie andernfalls den Eintrag aus der Tabelle (bzw. aus der verketten Liste innerhalb eines Buckets). Aktualisieren Sie die interne Buchführung und geben Sie `true` zurück.

j) Mit Hilfe der Methode `resize()` kann die Anzahl der Buckets der Streutabelle erhöht werden. Reinitialiseren Sie hierzu enstprechend die Buckets sowie die interne Buchführung. Fü-

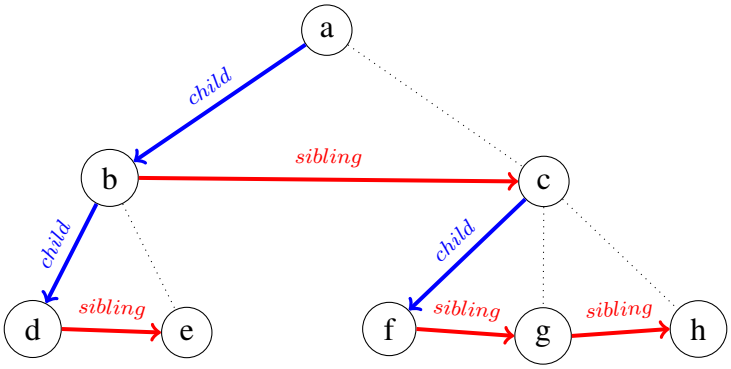
gen Sie anschließend die bereits vorhandenen Einträge wieder in die Streutabelle ein. *Wichtig:* Fügen Sie die Einträge in der aufsteigenden Reihenfolge der Indizes des `buckets`-Arrays erneut hinzu. Falls es in einem Bucket eine Liste von Einträgen gibt, dann müssen diese in der Reihenfolge der Liste erneut eingefügt werden (bevor mit dem nächsten Bucket fortgefahren wird).

k) Lesen Sie die Javadoc-Kommentare in der Schnittstellendefinition. Möglicherweise müssen Sie manche der von Ihnen implementierten Methoden noch durch Fehlerbehandlung erweitern.

Hiermit ist die Implementierung der Streutabellenfunktionalität abgeschlossen. Geben Sie Ihre Lösung als `ResizingHashMap.java` über das EST ab. Zum Testen ihrer Implementierung können sie Testfälle von der AuD-Seite herunterladen: ResizingHashMapPublicTest.

Gruppenaufgabe 10.4: Binärisierte Bäume — 23 GP

Beliebige Gewurzelte Bäume kann man mit Hilfe einfacher Knoten für Binärbäume darstellen, indem man den Zeiger für das sonst rechte Kind als Zeiger auf den jeweils nächsten Geschwisterknoten („*sibling*“) der gleichen Ebene verwendet, wodurch die Kinder eines Knotens als verkettete Liste auftreten:



a) Laden Sie die vorgegebene abstrakte Klasse `AbstractBinNode` von der AuD-Homepage und erstellen Sie eine Unterklasse davon namens `BinNode`.

b) Implementieren Sie nun die fehlenden Methoden anhand der jeweiligen Kommentare in der abstrakten Oberklasse bzw. in der zugehörigen API-Dokumentation. Bäume ohne Sortierung der Knotenwerte dürfen `null` als `value` enthalten – alle anderen Baumarten sind jedoch auf die natürliche Ordnung angewiesen, daher ist hier `null` für derartige Bäume verboten.

Geben Sie Ihre Lösung als `BinNode.java` über EST ab.

37 EP + 23 GP = 60 Punkte