

Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger
Frühjahrssemester 2018

Universität Basel
Fachbereich Informatik

Übungsblatt 7

Abgabe: 27. April 2018

Für dieses Übungsblatt benötigen Sie eine Java Umgebung, wie in Übungsblatt 1 beschrieben.
Falls sie keine IDE verwenden, können Sie ihre Programme von der Kommandozeile mittels

```
.\gradlew redBlackBSTExperiments  
.\gradlew hashtableExperiments
```

starten.

Das kompilieren und ausführen der Testfälle erfolgt wie in den vorhergehenden Übungen durch `gradlew build` respektive `gradlew test`.

Als Grundlage für Ihre Implementation verwenden Sie das Projekt im Zip-File "blatt07.zip", welches Sie auf dem Adam-Workspace finden. Die Lösungen zu den Theoriefragen geben Sie als Dokument im PDF Format, mit dem Namen `loesung-7.pdf` ab.

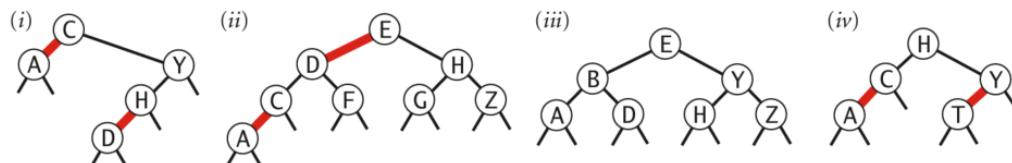
Bitte fügen sie diese PDF Datei zur Projektstruktur hinzu. Laden Sie Ihre Lösungen als zip Datei auf Courses hoch.

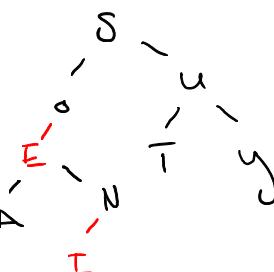
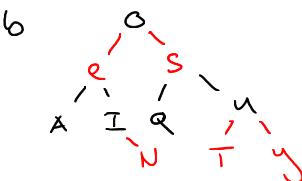
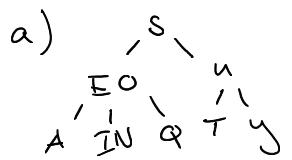
Aufgabe 7.1 (Rot-Schwarz Bäume, 2 Punkte)

- Zeichnen Sie den 2-3-Baum und den Rot-Schwarz Baum, den Sie erhalten, wenn Sie die Buchstaben E A S Y Q U T I O N in genau dieser Reihenfolge in einen anfänglich leeren Baum einfügen.
- Wahr oder falsch:
 - Wenn Sie Schlüssel in aufsteigender Reihenfolge in einen RotSchwarz-Baum einfügen, steigt die Baumhöhe monoton.
 - Wenn Sie Schlüssel in absteigender Reihenfolge in einen RotSchwarz-Baum einfügen, steigt die Baumhöhe monoton.
- Finden Sie eine Einfügereihenfolge für die Schlüssel S E A R C H X M , die zu einem 2-3-Baum der Höhe 1 führt.
- Welche der folgenden Strukturen sind gültige Rot-Schwarz-Bäume?

Tip: Sie können ihre Antworten prüfen indem Sie das Jupiter Notebook (`rbtree.ipynb`) verwenden, welches Sie auf der Kurswebseite

<http://informatik.unibas.ch/fs2018/vorlesung-algorithmen-und-datenstrukturen/>
finden.





b) wahr
falsch

gemäß Internet

gemäß Buch

c) SEARCHXM
EAXCHRMS

d) III & IV

I nicht balanced

II nicht sym + balanced

7.3) a)

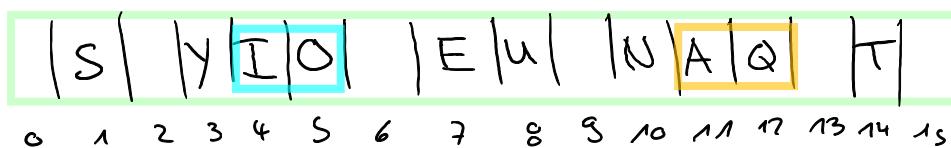
	E	A	S	Y	Q	U	T	I	O	N
numVal	S	1	19	25	17	21	20	9	15	14
.	11	55	11	205	275	187	231	220	89	165
%	5	0	1	4	0	2	1	0	4	0
	0	1	4	0	2	1	0	4	0	4

=>

- | | |
|---|---------|
| 0 | 0-T-Y-E |
| 1 | U-A |
| 2 | Q |
| 3 | Null |
| 4 | N-I-S |

b)

	E	A	S	Y	Q	U	T	I	O	N
numVal	S	1	19	25	17	21	20	9	15	14
.	11	55	11	205	275	187	231	220	89	165
%	16	7	11	1	3	11	7	14	3	5



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Aufgabe 7.2 (Vergleiche in Bäumen, 1 Punkt)

Überarbeiten Sie den Code RedBlackBST und BST (im package `edu.princeton.cs.algs4`) so, dass sie bei jeder `put` Operation die Anzahl Vergleiche gezählt werden. Implementieren Sie dann die Methode `RedBlackBSTExperiment.plotNumberOfComparisons`. Diese Methode sollen N zufällig generierte Schlüssel in einen anfangs leeren Baum Rot-Schwarz Baum, respektive Binären Suchbaum einfügen und die Anzahl der benötigten Vergleiche für $N = 0, \dots, 100000$ plotten. Diskutieren Sie das Ergebnis.

Aufgabe 7.3 (Einfügen in Hashtabelle, 1 Punkt)

- Fügen Sie die Buchstaben E A S Y Q U T I O N in genau dieser Reihenfolge in eine anfänglich leere Tabelle der Größe $M = 5$ ein. Verwenden Sie dabei Hashing mit Verkettung. Wandeln Sie mit der Hashfunktion $11k \% M$ den k -ten Buchstaben des Alphabets in einen Tabellenindex um. Zeichnen sie die resultierende Tabelle auf.
- Fügen Sie die Buchstaben E A S Y Q U T I O N in genau dieser Reihenfolge in eine anfänglich leere Tabelle der Größe $M = 16$ ein. Verwenden Sie dabei Lineare Sondierung. Wandeln Sie mit der Hashfunktion $11k \% M$ den k -ten Buchstaben des Alphabets in einen Tabellenindex um. Zeichnen sie die resultierende Tabelle auf.

Aufgabe 7.4 (Anwendung von Hashtabellen, 2 Punkte)

- Implementieren Sie die Klasse `exercise7.MathSet` mithilfe der Java Hashtable `java.util.Hashtable`.
- Testen Sie Ihre Implementation mittels der entsprechenden Unittests.

Aufgabe 7.5 (Hashing Methoden, (2 + 2) Punkte)

Sie finden im Projekt die Datei `LinearProbingHashST.java`, welche eine Implementation einer Hashtabelle mit Linearer Sondierung zur Konfliktresolution beinhaltet.

- Implementieren Sie die Methode `getMeanAndVarianceOfClusterLengths`, welche den Mittelwert und die Variance von den Clustern (d.h. aufeinanderfolgende, nicht leere Werte in der Hashtabelle) berechnet. Testen Sie Ihre Implementation mit den entsprechenden Unittests (Sie finden diese in `HashtableTests.java`). Erklären Sie, was diese Zahlen (also Durchschnitt und Varianz der Clusterlänge) für eine praktische Relevanz haben.
- *Double hashing* ist eine etwas raffiniertere Methode zum Auflösen von Konflikten. Informieren Sie sich wie diese Methode funktioniert (zum Beispiel bei Wikipedia) und implementieren Sie diese. Dazu müssen Sie die Methoden `get`, `put`, `resize` in der Klasse `DoubleHashingST` implementieren. Orientieren Sie sich dabei an der Implementation der entsprechenden Methoden von `LinearProbingHashST`. Stellen Sie auch sicher, dass die Grösse der Tabelle immer einer Primzahl entspricht. Führen Sie Experimente aus und diskutieren Sie wie sich die Clusterlänge und Varianz gegenüber der linearen Sondierung unterscheidet. Implementieren Sie Ihre Experimente in der Datei `HashtableExperiments.java`.

Diskussion: Machst du IntelliJ, kuckst du selbst.

→ mean: gleich \Rightarrow im schneller Cluster gleich gross [lin / double]

→ Var: bei double halb \Rightarrow weniger Streung.

↳ bei doubleHash Abweichung kleiner
somit cluster grösser bei jedem kleiner