Datenstrukturen und effiziente Algorithmen I

Sommersemester 2023
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Lehrstuhl für Datenstrukturen und effiziente Algorithmen

Übungsblatt 12

Abgabe: 04. Juli 2023

Aufgabe 1: Aufbau und Löschen im Suchbaum (2+2 Punkte)

a) Fügen Sie nacheinander die folgenden Schlüssel in einen anfangs leeren binären Suchbaum ein und zeichnen Sie den fertigen Baum.

32, 28, 50, 40, 20, 63, 29, 25, 17, 57, 37, 42, 23, 24

b) Löschen Sie nun nacheinander die folgenden Schlüssel aus dem Suchbaum und zeichnen Sie den fertigen Baum.

17, 20, 28, 50, 40

Aufgabe 2: Eigenschaften von Suchbäumen (3 Punkte)

Einer Ihrer Kommilitonen glaubt, eine schöne Eigenschaft binärer Suchbäume gefunden zu haben: Die Suche nach dem Schlüssel k ende in einem Blatt und man betrachte die folgenden drei Mengen: L ist die Menge der Schlüssel, die links vom Suchpfad liegen, P die Menge der Schlüssel auf dem Suchpfad und R die der Schlüssel rechts vom Suchpfad. Nun behauptet er, dass jedes Tripel von Schlüsseln $\ell \in L$, $p \in P$ und $r \in R$ die Ungleichung $\ell \leq p \leq r$ erfüllen muss. Stimmt das? Beweisen Sie die Richtigkeit dieser Aussage oder widerlegen Sie sie!

Aufgabe 3: Implementierung: Suchbaum (7+3 Punkte)

a) Implementieren Sie einen binären Suchbaum wie beschrieben auf Folie 294f. Ihre Klasse soll folgende Methoden bieten:

- `TreeNode<KeyType, DataType> search(KeyType k)`: Siehe Folie 295.
- `DataType isMember()`: Siehe Folie 296.
- `boolean insert(DataType d, KeyType k)`: Siehe Folie 299.
- `boolean remove(KeyType k)`: Siehe Folie 300f.
- `TreeNode<KeyType, DataType> minimum()`: Gibt den Baumknoten mit dem kleinsten Schlüssel zurück.

- `TreeNode<KeyType, DataType> maximum()`: Gibt den Baumknoten mit dem größten Schlüssel zurück.
- `void clear()`: Löscht alle Elemente.
- `int size()`: Bestimmt rekursiv die Anzahl der Elemente.
- `int depth()`: Bestimmt rekursiv die Tiefe des Baumes (die Anzahl der Kanten auf dem längsten Pfad von der Wurzel zu einem Blatt)
- `String toString()`: Gibt alle Schlüssel-Werte-Paare in Pre-Order (WLR) zeilenweise als String zurück.

Bonus `String toString(KeyType lb, KeyType ub)`: Gibt alle Schlüssel-Werte-Paare, deren Schlüssel in dem Intervall $[lb, ub]$ liegen, zeilenweise als String zurück.

Hinweise: Orientieren Sie sich bei der Implementierung der Methoden `search`, `isMember`, `insert` und `remove` an den Vorgaben aus der Vorlesung. Beachten Sie, dass dass Ihr Programm kompilierbar sein soll, also müssen Sie insbesondere auch die Klasse `TreeNode` (F. 293) mit einsenden.

Testen Sie Ihre Implementierung!

b) Schreiben Sie eine Klasse `WordCount`, welche eine Textdatei einliest und für jedes Wort dieser Datei ihre Häufigkeit bestimmt. Ihr Programm soll dafür die Klasse `SearchTree` verwenden mit String als Schlüsseltyp und Integer als Wertetyp. Das Programm soll am Ende die 10 häufigsten Wörter mit den dazugehörigen Häufigkeiten ausgeben. Testen Sie Ihr Programm an der Datei De_Odyssee.txt. Ignorieren Sie Satzzeichen und Zahlen. Geben Sie die Ausgabe des Programmes als PDF mit ab.

c) Bonus (+3): Beschreiben Sie, wie eine Textdatei aussehen muss, bei der das Programm `WordCount` aus Aufgabe b) den schlechtest möglichen Gesamtaufwand hat und begründen Sie Ihre Antwort. Welche Tiefe hätte der Suchbaum in diesem Fall?

Aufgabe 4: Hashing mit Verkettung (3 Punkte)

Fügen Sie die Zeichen `B`, `C`, `G`, `I`, `X`, `A`, `Z` in eine Hashtabelle der Größe $m = 7$ ein.

(a) Überführen Sie dazu zunächst die Zeichen mittels einer ASCII-Tabelle in ganze Zahlen.

(b) Fügen Sie die Zeichen mittels der Divisionsmethode ein. Notieren Sie für jedes Zeichen die Hashposition und zeigen Sie die Veränderungen der Hashtabelle.

(c) Wiederholen Sie den Aufgabenteil (b) nun mittels der Multiplikationsmethode. Verwenden Sie dafür $a = 0,618$.