Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

8. Dezember 2016

Algorithmen & Datenstrukturen Lösungen zu Blatt P11 HS 16

Lösung P11.1 *Binäre Suchbäume.*

Die Lösung zu dieser Aufgabe wurde grob bereits in der Aufgabenstellung beschrieben. Die Details zur Implementierung finden Sie in der Beispiel-Implementierung auf der Webseite.

Grundsätzlich musste jede Operation einen gegebenen Wert mit dem Wert des momentan betrachteten Schlüssels `key` vergleichen und entweder den entsprechenden Knoten verändern, oder sich selbst rekursiv auf dem linken, bzw. rechten Teilbaum aufrufen. Die einzige Ausnahme war `delete` eines Knoten X mit zwei Kindern: Der symmetrische Nachfolger kann gefunden werden, indem man das rechte Kind von X betrachtet und von dort aus immer nach links geht bis man einen Knoten Y ohne linkes Kind erreicht. Dieser Knoten ist der symmetrische Nachfolger von X.

Um die Grösse der Teilbäume `size` zu aktualisieren, kann man entweder eine Funktion `node.fixSizesToRoot()` implementieren, welche die Grösse der Teilbäume entlang des Pfades von *node* zur Wurzel neu berechnet, oder die Grösse direkt in `insert` oder `delete` neu berechnen, da sich die Grösse nur um +1 oder -1 ändert (wie in der Aufgabenstellung erwähnt).

Die Grösse eines Teilbaumes zu aktualisieren ist ein einfaches Beispiel von *subtree aggregation*. Beispielsweise könnte man leicht auch die Summe oder das Produkt der Schlüssel eines Teilbaumes, die maximale Tiefe (wie bei AVL-Bäumen), den Durchschnittswert der Schlüssel, den kleinsten und den grössten Schlüsse, die Anzahl der geraden Schlüssel, etc. berechnen.

Lösungen Auf der Vorlesungswebseite finden Sie eine Lösung, welche weitere Kommentare zur Implementierung enthält.

Daten Die Tests `judge2` und `judge3` fügen – ähnlich wie die Tests `test*` – zuerst 30 000 verschiedene, zufällige Elemente so ein, dass die Tiefe des Baumes in etwa 30 beträgt. Dann werden weitere 10 000 zufällige Elemente eingefügt, wobei vor und nach dem Einfügen geprüft wird, ob das Element vorhanden ist. Der Test `judge3` fragt anschliessend den Rang von 10 000 Elementen ab, wobei sporadisch Elemente auch gelöscht werden, um zu verhindern, dass der Rang mithilfe eines Arrays berechnet wird. `judge1` ist nicht so umfangreich und enthält keine delete Operationen. Die Bäume werden in etwa fünf mal für jeden Testfall ausgegeben (da die Ausgabe eines Baumes lange dauert, wird diese Operation nur selten abgefragt).