Hochschule Fulda
University of Applied Sciences

Prof. Dr. Thomas Wiemann

Algorithmen und Datenstrukturen
Übungsblatt 8

Wintersemester 2024/25

Aufgabe 8.1 (Baumtraverse)

Traversieren Sie den fertigen Baum aus der vorherigen Aufgabe jeweils in Pre-, Post und In-Order-Folge. Geben Sie die resultieren Zahlenfolgen an. Was fällt auf?

Aufgabe 8.2 (Bestimmen der Baumhöhe)

Die Höhe eines Knotens v in einem binären Suchbaum T ist definiert als der kürzeste Pfad von diesem Knoten zu einem Blatt. Schreiben Sie einen Algorithmus mit der Laufzeit $\mathcal{O}(n)$, der für alle Knoten v in T ihre Höhe berechnet. Implementieren und testen Sie ihn in Java. Erklären Sie, warum Ihre Lösung lineare Laufzeit hat.

Aufgabe 8.3 (Sift)

Wenden Sie SIFT auf das folgende Array an (mit $i = 3$): [27,17,3,16,13,10,1,5,7,12,4,8,9,0]

Aufgabe 8.4 (Aufbau eines Max-Heaps)

Bauen Sie einen Max-Heap aus folgender Eingabe auf: [5,3,17,10,84,19,6,22,9].

Aufgabe 8.5 (Heap-Sort)

Führen Sie Heap-Sort auf folgendem Array schriftlich aus [5,3,17,10,84] . Zeigen Sie nach jedem Schritt den Zustand des Array und den dazugehörigen Heap.

Aufgabe 8.6 (Einfügen in einen Rot-Schwarz-Baum)

Unter `https://www.cs.usfca.edu/~galles/visualization/RedBlack.html` finden Sie das in der Vorlesung genutzte Applet zur Visualisierung des Balancierens von Rot-Schwarz-Bäumen. Fügen Sie die Zahlen von 1 bis 10 nacheinander ein und erklären Sie die entstehenden Operationen anhand der RBT-Regeln aus der Vorlesung.

Aufgabe 8.7 (Rot-Schwarz-Baum - Rechtsrotation)

Der folgende Code, der dazu dient, eine „Rechtsdrehung“ in einem Rot-Schwarz-Baum (mit linksgerichteten roten Verknüpfungen) zu implementieren. Diese Operation ist für alle Einfügungen in den Baum erforderlich enthält einen Fehler. Beschreiben Sie diesen Fehler und seine Behebung. Anmerkung: das Feld `size` repräsentiert die Anzahl der Knoten im jeweiligen Unterbaum und kann bei der Betrachtung ignoriert werden.

```
private Node rotateRight(Node n) {
  Node t = n.left;
  n.left = t.right;
  t.right = n;
  t.color = n.color;
  t.size = n.size;
  n.size = 1 + size(n.left) + size(n.right);
  return t;
}
```

Aufgabe 8.8 (Rot-Schwarz-Baum - Einfügen)

Betrachten Sie den folgenden Code, der ein neues Schlüssel-Wert-Paar in einen Rot-Schwarz-Baum einfügen soll. Nehmen Sie an, dass die Klasse „Node“ angesichts ihrer Verwendung unten implementiert ist und die Methoden „rotateLeft()“, „rotateRight()“ und „flipColors()“ alle wie erwartet funktionieren. Dennoch enthält dieser Code einen Fehler: Die Zeilen mit den Kommentaren „Zeile 1“, „Zeile 2“ und „Zeile 3“ sind nicht in der richtigen Reihenfolge. In welcher Reihenfolge sollten sie angeordnet werden? Anmerkung: die Felder `count` und `size` sollen die Anzahl der Knoten im jeweiligen Unterbaum repräsentieren und können bei der Betrachtung ignoriert werden.

```
private Node insert(Node n, Key key, Value val) {
   if (n = null) {
      Node newNode = new Node(key, val, RED);
      newNode.count = 1;
      return newNode;
   }

   int cmp = key.compareTo(n.key);
   if (cmp < 0)         n.left = put(n.left, key, val);
   else if (cmp > 0)    n.right = put(n.right, key, val);
   else                 n.val = val;

   if (isRed(n.left) && isRed(n.right)       flipColors(n);         \\# Line 1
   if (isRed(n.right) && !isRed(n.left))     n = rotateLeft(n);     \\# Line 2
   if (isRed(n.left) && isRed(n.left.left))  n = rotateRight(n);    \\# Line 3

   n.count = 1 + size(n.left) + size(n.right);
   return n;
}
```