Algorithmen und Datenstrukturen
Musterlösung 5

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

12. April, zur Besprechung am 3. Mai

Aufgabe 1) Amortisierte Kosten Die while-Schleife wird vor allem dann oft durchlaufen, wenn `bits` viele 1en (am Ende) enthält. Diese 1en werden dann aber alle in 0en verwandelt. Deshalb definieren wir G durch $G(\texttt{bits}) = \#1\text{en in } \texttt{bits}$.

Es ist nun einfach einzusehen, dass jeder Schleifendurchlauf, der nicht zum Abbruch führt, amortisierte Kosten von 0 hat: amort. Kosten = reale Kosten + $(G_{nachher} - G_{vorher}) = 1 + (-1) = 0$. Wenn die Schleife abgebrochen wird, erhält man amortisierte Kosten von 1, da G unverändert bleibt. Folglich sind die amortisierten Kosten der Abarbeitung der while-Schleife genau $a_{while} = 1$.

Danach lassen sich die gesamten amortisierten Kosten wie folgt abschätzen:

$$a_{total} = a_9 + a_{while} + a_{18-20} = 1 + 1 + a_{18-20} \leq 1 + 1 + 2 = \mathcal{O}(1)$$

Die amortisierten Kosten von Zeilen 18-20 setzen sich aus den realen Kosten von 1 und einer möglichen Guthabendifferenz von 1 zusammen.

Aufgabe 2) Einfügen und Löschen in AVL Bäume

1. Die Lösung für Teilaufgabe a) und b) kann mit `http://www.cp.eng.chula.ac.th/~vishnu/datastructure/AVL/AVL-Applet.html` oder der Lösung zu Aufgabe 3 interaktiv nachgeprüft werden.
2. Teilaufgabe c) - es ist nicht möglich diesen Wert aus den Kindknoten abzuleiten. Der `hdiff` muss im Verlauf von Modifikationen entsprechend angepasst werden.
3. `n.hdiff = (r.hdiff==1)?-1:0` und `p.hdiff = (r.hdiff==-1)?1:0`. Der `hdiff` Wert für Knoten r kann vor der Rotation nur -1 oder 1 sein. Durch Fallunterscheidung ergeben sich die jeweiligen Ergebnisse.

Aufgabe 3) AVL Implementierung

- Suchen eines Elements:

```java
/**
 * Tests whether this AVL tree contains the given element.
 *
 * @param element
 *            the element to be searched
 * @return true if contained, false otherwise
 */
public boolean contains(int element) {
    // start at the root node
    Node cur = root;
    while (cur != null) {
        // check current node
        if (element == cur.value) {
            return true;
        }

        // continue search within proper sub-tree
        if (element < cur.value) {
            cur = cur.left;
        } else {
            cur = cur.right;
        }
    }

    // not found => not contained
    return false;
}
```

- Einfügen + Rotieren + Balancieren:

```java
/**
 * Inserts the given element inside this tree.
 *
 * @param value
 *            the value to be inserted
 */
public void insert(int value) {
    // handle empty tree
    if (root == null) {
        root = new Node(value, null);
        return;
    }

    // search insertion place
    Node p = root;

    // search insertion point and add node
    while (true) {
        Node next = (value < p.value) ? p.left : p.right;
        if (next != null) {
            p = next;
        } else {
            break;
        }
    }

    // add new node
```

```
28          Node newNode = new Node(value, p);
29          if (value < p.value) {
30              // one more node on the left side
31              p.left = newNode;
32              p.hdiff--;
33          } else {
34              // one more node on the right side
35              p.right = newNode;
36              p.hdiff++;
37          }
38
39          // if size of sub-tree has changed ...
40          if (p.hdiff != 0) {
41              // ... correct balancing => balGrow
42              balGrow(p);
43          }
44      }
45
46      /**
47       * React on the grew of the given sub-tree.
48       *
49       * @param n
50       *             the root node of the sub-tree which grew by one
             level.
51       */
52      private void balGrow(Node n) {
53          assert (n.hdiff == -1 || n.hdiff == 1);
54
55          // get parent node
56          Node p = n.parent;
57          if (p == null) {
58              // n is root => nothing to do
59              return;
60          }
61
62          // Now every case has to be evaluated ...
63
64          // test whether left or right sub-tree grew
65          if (p.left == n) {
66              if (p.hdiff == 1) {
67                  // balanced => done
68                  p.hdiff = 0;
69                  return;
70              } else if (p.hdiff == 0) {
71                  // subtree grew => continue with parent
72                  p.hdiff = -1;
73                  balGrow(p);
74                  return;
75              } else if (n.hdiff == -1) { // and p.hdiff = -1
76                  rotateRight(p);
77                  p.hdiff = 0;
78                  n.hdiff = 0;
79                  return;
80              } else { // n.hdiff = 1, p.hdiff = -1
81                  // double rotation
82                  Node r = n.right;
83                  rotateLeft(n);
84                  rotateRight(p);
85                  n.hdiff = (r.hdiff == 1) ? -1 : 0;
86                  p.hdiff = (r.hdiff == -1) ? 1 : 0;
87                  r.hdiff = 0;
88                  return;
```

```
89              }
90          } else { // p.right = n
91              if (p.hdiff == -1) {
92                  // balanced => done
93                  p.hdiff = 0;
94                  return;
95              } else if (p.hdiff == 0) {
96                  // subtree grew => continue with parent
97                  p.hdiff = 1;
98                  balGrow(p);
99                  return;
100             } else if (n.hdiff == 1) { // and p.hdiff = 1
101                 rotateLeft(p);
102                 p.hdiff = 0;
103                 n.hdiff = 0;
104                 return;
105             } else { // n.hdiff = -1, p.hdiff = 1
106                 // double rotation
107                 Node l = n.left;
108                 rotateRight(n);
109                 rotateLeft(p);
110                 n.hdiff = (l.hdiff == -1) ? 1 : 0;
111                 p.hdiff = (l.hdiff == 1) ? -1 : 0;
112                 l.hdiff = 0;
113                 return;
114             }
115         }
116     }
117
118     /**
119      * This method is applying a left-rotation on the given node. His
            right
120      * child will become the new root node of the corresponding sub-
            tree.
121      *
122      * @param n
123      *             the node to be rotated toward the left.
124      */
125     private void rotateLeft(Node p) {
126         Node n = p.right;
127
128         // update parents
129         n.parent = p.parent;
130         p.parent = n;
131
132         // update right
133         p.right = n.left;
134         if (p.right != null) {
135             p.right.parent = p;
136         }
137
138         // update left
139         n.left = p;
140
141         // also: update original parent of p (potentially the root
            node)
142         Node gp = n.parent;
143         if (gp == null) {
144             // p was the root node => now it is n
145             root = n;
146         } else if (gp.left == p) {
147             // p was the left child
```

```
148             gp.left = n;
149         } else {
150             // p was the right child
151             gp.right = n;
152         }
153     }
154 
155     /**
156      * This method is applying a right-rotation on the given node. His
            left
157      * child will become the new root node of the corresponding sub-
            tree.
158      *
159      * @param n
160      *             the node to be rotated toward the right.
161      */
162     private void rotateRight(Node p) {
163         Node n = p.left;
164 
165         // update parents
166         n.parent = p.parent;
167         p.parent = n;
168 
169         // update left
170         p.left = n.right;
171         if (p.left != null) {
172             p.left.parent = p;
173         }
174 
175         // update right
176         n.right = p;
177 
178         // also: update original parent of p (potentially the root
                node)
179         Node gp = n.parent;
180         if (gp == null) {
181             // p was the root node => now it is n
182             root = n;
183         } else if (gp.left == p) {
184             // p was the left child
185             gp.left = n;
186         } else {
187             // p was the right child
188             gp.right = n;
189         }
190     }
```