RWTH Aachen **SS 2015**
Lehrgebiet Theoretische Informatik Blatt 3
Rossmanith–Reidl–Sánchez 29. April 2015

Übung zur Vorlesung Algorithmen und Datenstrukturen

Aufgabe T7

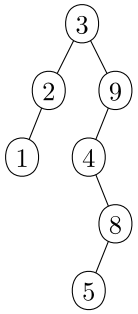
In einem Splay-Baum können wir so nach einem Schlüssel suchen:

```
public boolean containsKey(K k) {
  if(root == null) return false;
  Searchtreenode⟨K, D⟩ n = root, last = root;
  int c;
  while(n ≠ null) {
    last = n;
    c = k.compareTo(n.key);
    if(c < 0) n = n.left;
    else if(c > 0) n = n.right;
    else { splay(n); return true; }
  }
  splay(last); return false;
}
```

Warum gibt es am Ende die Anweisung *splay*(*last*)? Überlegen Sie sich ein Beispiel, in welchem bei einem anfangs leeren Splay-Baum n Operationen zu einem Zeitaufwand von $\Theta(n^2)$ führen, falls diese Anweisung fehlt.

Aufgabe T8

Wir betrachten folgenden Splay-Baum:



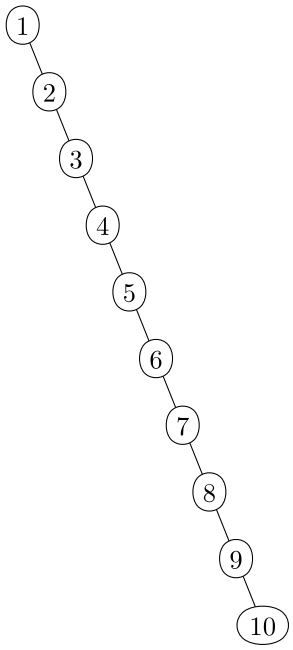
Was passiert, wenn wir in diesen Baum nach dem Schlüssel 10 suchen, dann nach 1 suchen, dann 6 einfügen und schließlich 8 löschen?

Aufgabe T9

Entwerfen Sie einen effizienten Algorithmus, der einen Splay-Baum B und einen Schlüssel k bekommt und den Baum B in zwei Splay-Bäume B_1 und B_2 teilt, wobei B_1 alle Schlüssel enthält, die kleiner oder gleich k sind und B_2 die übrigen Schlüssel.

Aufgabe H5

Wir betrachten folgenden Splay-Baum, der recht unbalanciert ist:



Was passiert, wenn wir erst nach dem Schlüssel 10 suchen und dann nach dem Schlüssel 9? Ist der Baum jetzt besser balanciert?

Aufgabe H6

Entwerfen Sie einen effizienten Algorithmus, der im wesentlichen das Gegenteil vom Algorithmus der Aufgabe T9 vollführt: Als Eingabe erhält er zwei Splay-Bäume B_1 und B_2, wobei garantiert wird, daß kein Schlüssel in B_2 kleiner ist, als ein Schlüssel in B_1. Als Ausgabe soll er einen einzigen Splay-Baum B erzeugen, der nun alle Schlüssel von B_1 und B_2 enthält.