Grundzüge von Algorithmen und Datenstrukturen, WS 15/16: Lösungshinweise zum 6. Übungsblatt

Markus Bläser

Aufgabe 6.1 (Keine Lösung vorhanden)

Aufgabe 6.2 (Lösungshinweise) Wir machen einen Inorder-Durchlauf, steigen aber in einen Teilbaum nicht ab, wenn der Schlüssel $< k_1$ oder $> k_2$ ist:

Interval
Input: A node x of a binary search tree T, keys k_1, k_2
Output: prints the elements of $T(x)$ in increasing key order with keys between k_1 and k_2

```
1: if x ≠ NULL then
2:     if Key(x) ≥ k_1 then
3:         Interval(Left(x), k_1, k_2)
4:     if k_1 ≤ Key(x) ≤ k_2 then
5:         output x
6:     if Key(x) ≤ k_2 then
7:         Interval(Right(x), k_1, k_2)
```

Den Beweis der Korrektheit führt man mit struktureller Induktion. Wir zeigen: Interval(x, k_1, k_2) gibt alle Elemente mit Schlüssel zwischen k_1 und k_2 in $T(x)$ aus. Das ist bestimmt richtig für den leeren Baum. Induktionsschritt: Nach Induktionsannahme werden in Zeile 2–3 alle Elemente von $T(\text{Left}(x))$ ausgegeben, deren Schlüssel in Bereich $k_1, \ldots, k_2$ liegen. Wegen der binären Suchbaumeigenschaft muss der linke Teilbaum nur betrachtet werden, wenn der Key von x mindestens k_1 ist. Die gleiche Argumentation gilt für den rechten Teilbaum in Zeile 6–7. Da alle Elemente entweder im linken Teilbaum oder im Knoten x selbst oder im rechten Teilbaum sind, gibt der Algorithmus alle Elemente in $T(x)$ mit Schlüsseln im Bereich $k_1, \ldots, k_2$ aus.

Laufzeit: Die Laufzeit ist $O(h + \ell)$, wobei h die Höhe des Baumes ist und ℓ die Länge des ausgegebenen Intervalls. Dies sieht man wie folgt: Bis das erste Element mit Schlüssel k_1 gefunden wird, verhält sich die Prozedur wie BST-Search. Danach gibt sie in jedem Schritt ein Element aus, bis sie auf das nächstgrößere Element nach k_2 stößt. Dann wird abgebrochen.

Aufgabe 6.3 Sei T der binäre Suchbaum (mit n Knoten), in den die Pivotelemente entsprechend der Reihenfolge ihres Auftretens in allen Quicksort-Aufrufen eingefügt

wurden. Als *Gewicht* eines Teilbaums bezeichnen wir dir Anzahl seiner Knoten. Es gilt: jeder einzelne Quicksort-Aufruf x braucht Zeit proportional zum Gewicht des Teilbaums von T, dessen Wurzeln p_x ist. Denn unterhalb von p_x sind genau die Elemente gespeichert, die im Aufruf x betrachtet werden.

Wenn T Tiefe cn hat für ein $c > 0$, dann gibt es einen Pfad von der Wurzel zu einem Blatt mit Tiefe cn. Die Summen der Gewichte der Teilbäume ausgehend von diesem Blatt (das heißt: vom Blatt auf die Wurzel zu) ist dann mindestens

$$1 + 2 + \ldots + \lfloor cn \rfloor \leq cn(cn+1)/2 \in \Omega(n^2),$$

da die Gewichte der Teilbäume jeweils um mindestens einen Knoten zunehmen. Andererseits gilt: Da wir n Knoten haben und jeder Teilbaum von T höchstens n Knoten enthält, ist die Summe der Gewichte der Teilbäume höchstens n^2.

Habe nun T logarithmische Tiefe, das heißt, es gibt ein $c > 0$, so dass die Tiefe von T höchstens $c \log n$ ist. Wir betrachten alle Knoten, die sich in einer Ebene befinden. Seien dies die Knoten $v_1, \ldots, v_r$. Die Teilbäume von T, die diese Knoten als Wurzel haben, seien $T_{v_1}, \ldots, T_{v_r}$. Diese Teilbäume sind disjunkt. Also haben sie zusammen höchstens n Knoten. Damit gilt:

$$\sum_{v \text{ Knoten in } T} |T_v| \leq \sum_{0 \leq i \leq c \log n} \sum_{0 \leq j \leq r(i)} |T_{v_j^{(i)}}| \leq \sum_{0 \leq i \leq c \log n} n = cn \log n,$$

wobei $r(i)$ für die Anzahl der Knoten in Ebene i steht und $v_1^{(i)}, \ldots, v_{r(i)}^{(i)}$ die Knoten der Ebene i bezeichnen.