Institut für Informatik I
Prof. Dr. Heiko Röglin
Tobias Brunsch

Algorithmen und Berechnungskomplexität I
Wintersemester 2010/11
Abgabe: 03.11.2010 in der Vorlesung

Übungsblatt 3

Aufgabe 3.1

1+2+3 Punkte

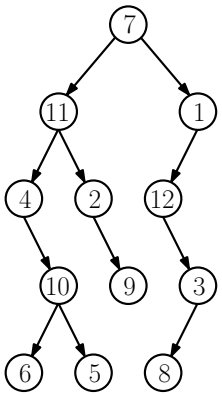
Wir betrachten die folgende rekursive Funktion `Inorder`(T), die als Parameter einen binären Baum T erhält.

```
Inorder(T)
{
  if T = leer then return
  Inorder(T_l)            // T_l ist der linke Teilbaum von T
  Gib Key_T aus.          // Key_T ist der Schlüssel des Wurzelknotens von T
  Inorder(T_r)            // T_r ist der rechte Teilbaum von T
}
```

(a) Was gibt die Funktion `Inorder` aus, wenn sie den rechts abgebildeten binären Baum als Eingabe erhält?

(b) Geben Sie an, wie man mit Hilfe der Funktion `Inorder` und ggf. weiteren Datenstrukturen einen Sortieralgorithmus mit Laufzeit $O(n \log n)$ konstruieren kann. Begründen Sie, warum Ihre Idee funktioniert.

(c) Beim Einfügen in einen binären Suchbaum ist die Reihenfolge der Schlüssel entscheident. Beim Löschen ist die Relevanz der Reihenfolge nicht sofort ersichtlich. Geben Sie einen binären Suchbaum T und zwei Schlüssel k_1 und k_2 an, für die gilt: Der Baum T_1, den man erhält, wenn man erst k_1 und dann k_2 aus T löscht, unterscheidet sich von dem Baum T_2, den man erhält, wenn man erst k_2 und dann k_1 aus T löscht.



Aufgabe 3.2

4+2 Punkte

(a) Fügen Sie die Schlüssel $1, \ldots, 10$ in dieser Reihenfolge in einen zu Beginn leeren B-Baum der Ordnung $t = 2$ ein. Zeichnen Sie den Baum nach jeder Operation.

(b) Löschen Sie anschließend die Schlüssel $8, 4, 9$ in dieser Reihenfolge. Zeichnen Sie den Baum nach jeder Operation.

Aufgabe 3.3

6 Punkte

Ein B^*-Baum der Ordnung t ist ein B-Baum der Ordnung t, bei dem jeder Knoten außer der Wurzel nicht nur mindestens $t-1$ Schlüssel, sondern mindestens $\left\lfloor \frac{4t-2}{3} \right\rfloor$ Schlüssel enthält. Beschreiben Sie eine `Insert`-Operation für B^*-Bäume. Konzentrieren Sie sich dabei auf die wesentliche Idee. Quelltext ist nicht nötig.

Hinweis: Die `Insert`-Operation von B-Bäumen, bei der ein Knoten aufgespaltet wird, wenn er voll ist, kann obige Eigenschaft nicht gewährleisten. Versuchen Sie, das Aufspalten eines Knotens solange hinauszuzögern, bis auch einer seiner Nachbarn voll ist.

Aufgabe 3.4

3+3 Punkte

(a) Gegeben sei ein Feld der Größe $m = 13$, das als Hashtabelle mit den Indizes $0, \ldots, m-1$ dient. Wir betrachten Hashing mit verketteten Listen, d.h. jeder Eintrag des Feldes enthält einen Zeiger auf eine (zu Beginn leere) verkettete Liste. Fügen Sie die Schlüssel

$$1207, 286, 1195, 298, 62, 573, 623, 6280, 89, 199, 447$$

in der angegebenen Reihenfolge gemäß der Hashfunktion $h(x) := x \mod m$ in die Hashtabelle ein. (Es genügt, das Endergebnis aller Einfügeoperationen anzugeben.)

(b) Gegeben seien n paarweise verschiedene Schlüssel $x_1, \ldots, x_n$ und eine Tabelle mit m Einträgen (m und n beliebig). Wir wählen sukzessive für jeden Schlüssel x_i zufällig eine Position p_i der Tabelle, auf die x_i abgebildet wird. Dabei soll für jedes x_i jede Position $0, \ldots, m-1$ gleichwahrscheinlich und unabhängig von der Wahl der Positionen $p_1, \ldots, p_{i-1}$ sein. Als Kollision bezeichnen wir wieder die Situation, dass zwei verschiedene Schlüssel auf dieselbe Position abgebildet werden. Bestimmen Sie die erwartete Anzahl an Kollisionen. Die Anzahl an Kollisionen sei dabei definiert als die Anzahl aller Paare (i, j) mit $1 \leq i < j \leq n$ und $p_i = p_j$.

Hinweis: Nutzen Sie Linearität des Erwartungswertes.