Johannes Blömer
u.v.a.m.

Paderborn, den 27. Juni 2004

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 9

AUFGABE 1:

1. Beim linearen Hashing gilt
$$h(k,i) = h'(k) + i.$$
Das Objekt x hat Schlüssel $k = 15$ und es gilt $h'(15) = 4$. Damit gilt
$$h(k,0) = 4, h(k,1) = 5 \quad \text{und} \quad h(k,2) = 6.$$
Da $T[4]$ und $T[5]$ bereits belegt sind, Feld $T[6]$ hingegen frei ist, werden also bei Einfügen von x die Positionen $4, 5$ und 6 getestet.

2. Beim doppelten Hashen gilt
$$h(k,i) = h_1(k) + ih_2(k).$$
Für den Schlüssel $k = 15$ gilt
$$h_1(15) = 4 \quad \text{und} \quad h_2(15) = 6.$$
Damit gilt
$$h(k,0) = 4 \quad \text{und} \quad h(k,2) = 10.$$
Da $T[4]$ bereits belegt ist, Feld $T[10]$ hingegen frei ist, werden also bei Einfügen von x die Positionen 4 und 10 getestet.

AUFGABE 2:

Die Antworten sind in der folgenden Tabelle zusammengefasst.

	$\alpha = \frac{3}{4}$	$\alpha = \frac{7}{8}$
nicht erfolgreiche Suche	4	8
erfolgreiche Suche	$\frac{4}{3} \cdot \ln(4) \approx 1,8484$	$\frac{8}{7} \cdot \ln(8) \approx 2,3765$

AUFGABE 3:

Es seien höchstens n Elemente bereits in die Hashtabelle eingefügt. Mit X bezeichnen wir eine Zufallsvariable, die die Anzahl der erforderlichen Tests beim Einfügen eines weiteren Objekts zählt. Die gesuchte Wahrscheinlichkeit ist dann $\Pr(x > k)$. Damit wir mindestens $k + 1$ Tests benötigen, muss jeder der ersten k Tests auf eine besetzte Position sein. Beim offenen Hashing werden immer neue Positionen getestet (jede Testfolge ist eine Permutation von $(0, \ldots, m-1)$). Unter der Annahme des uniformen Hashings ist damit die Wahrscheinlichkeit, dass jeder der ersten k Tests auf eine besetzte Position stattfindet höchstens
$$\frac{n(n-1)\cdots(n-k+1)}{m(m-1)\cdots(m-k+1)}.$$
Hierbei gilt, dass die Wahrscheinlichkeit höchstens so groß ist, da wir nur annehmen, dass höchstens n Objekte bereits eingefügt wurden. Wir erhalten jetzt
$$\Pr(X > k) \leq \frac{n(n-1)\cdots(n-k+1)}{m(m-1)\cdots(m-k+1)} \leq \frac{n}{m}^k,$$
denn $\frac{n-j}{m-j} \leq \frac{n}{m}$ für $n \leq m$ und $j \geq 0$. Nach Annahme gilt $n \leq m/2$ und damit
$$\Pr(X > k) \leq 2^{-k}.$$

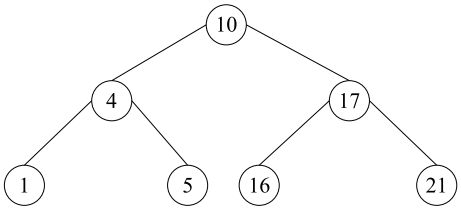

Abbildung 1: Suchbaum zu Aufgabe 4

AUFGABE 4:
Ein binärer Suchbaum für die Schlüsselmenge $\{1, 4, 5, , 10, 16, 17, 21\}$ ist in Abbildung 1 dargestellt.

AUFGABE 5:
Sei TREE-BUILD(S) ein *beliebiger* Algorithmus, der bei Eingabe einer Menge S von natürlichen Zahlen einen binären Suchbaum auf den Zahlen in S aufbaut. Dabei benutzte dieser Algorithmus nur die in der Aufgabenstellung genannten Operationen. Gegeben eine Menge von Zahlen S, können wir diese sortieren, indem wir zunächst Algorithmus TREE-BUILD(S) benutzen, um auf S einen binären Suchbaum T zu berechnen. Rufen wir dann Algorithmus INORDER-TREE-WALK mit Eingabe $root[T]$ auf, so erhalten wir die Zahlen in S in sortierter Reihenfolge. Sei T_1 die Laufzeit von TREE-BUILD(S), wenn S Größe n besitzt. Die Gesamtlaufzeit des gerade beschriebenen Sortieralgorithmus ist dann $\mathcal{O}(T_1(n) + n)$, denn wir haben in der Vorlesung gesehen, dass INORDER-TREE-WALK bei einem Baum mit n Knoten Laufzeit $\mathcal{O}(n)$ besitzt. Nun benötigt aber Algorithmus INORDER-TREE-WALK nur die in der Aufgabe genannten Operationen. Damit ist die Hintereinanderschaltung der Algorithmen TREE-BUILD und INORDER-TREE-WALK ein *Vergleichssortierer*. Wie wir in der Vorlesung gesehen haben, besitzen Vergleichssortierer Laufzeit $\Omega(n \log(n))$. Also muss auch $T_1(n) + n = \Omega(n \log(n))$ gelten. Dieses wiederum heisst $T(n) = \Omega(n \log(n))$.