

Beweis:

Die obere Schranke ist klar, da ein Binärbaum der Höhe h höchstens

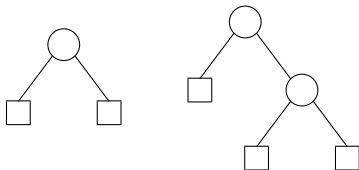
$$\sum_{j=0}^{h-1} 2^j = 2^h - 1$$

interne Knoten enthalten kann.

Beweis:

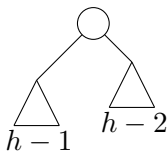
Induktionsanfang:

- 1 ein AVL-Baum der Höhe $h = 1$ enthält mindestens einen internen Knoten, $1 \geq F_3 - 1 = 2 - 1 = 1$
- 2 ein AVL-Baum der Höhe $h = 2$ enthält mindestens zwei Knoten, $2 \geq F_4 - 1 = 3 - 1 = 2$



Beweis:

Induktionsschluss: Ein AVL-Baum der Höhe $h \geq 2$ mit minimaler Knotenzahl hat als Unterbäume der Wurzel einen AVL-Baum der Höhe $h - 1$ und einen der Höhe $h - 2$, jeweils mit minimaler Knotenzahl.



Beweis:

Induktionsschluss: Ein AVL-Baum der Höhe $h \geq 2$ mit minimaler Knotenzahl hat als Unterbäume der Wurzel einen AVL-Baum der Höhe $h - 1$ und einen der Höhe $h - 2$, jeweils mit minimaler Knotenzahl. Sei

$f_h := 1 +$ minimale Knotenzahl eines AVL-Baums der Höhe h .

Dann gilt demgemäß

$$f_1 = 2 \qquad = F_3$$

$$f_2 = 3 \qquad = F_4$$

$$f_h - 1 = 1 + f_{h-1} - 1 + f_{h-2} - 1, \qquad \text{also}$$

$$f_h = f_{h-1} + f_{h-2} \qquad = F_{h+2}$$



Bemerkung:

Da

$$F(n) \approx \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n ,$$

hat ein AVL-Baum mit n internen Knoten eine Höhe

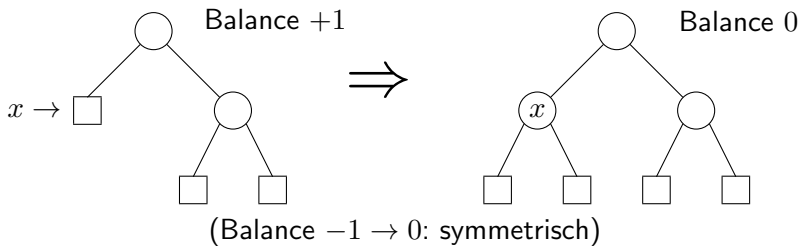
$$\Theta(\log n) .$$

Dies ist ein bedeutender Fortschritt gegenüber der bisweilen bei natürlichen Suchbäumen entstehenden, im *worst-case* linearen Entartung.

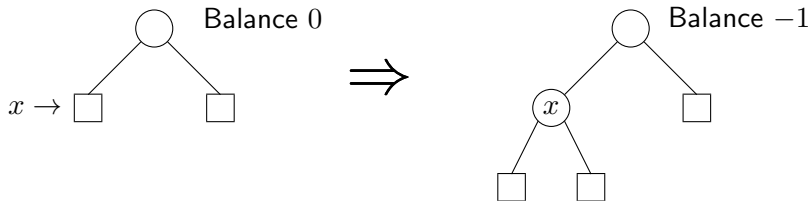
Die Operationen auf AVL-Bäumen:

- 1 *IsElement*: Diese Operation wird wie bei den natürlichen Suchbäumen implementiert. Wie oben ausgeführt, haben aber AVL-Bäume mit n Schlüsseln eine Höhe von $\mathcal{O}(\log n)$, woraus logarithmische Zeit für das Suchen folgt.
- 2 *Insert*: Zunächst wird eine *IsElement*-Operation ausgeführt, die für den Fall, dass das einzufügende Element nicht bereits enthalten ist, zu einem Blatt führt. Dies ist die Stelle, an der das neue Element einzufügen ist. Dabei ergeben sich 2 Fälle:

1. Fall:

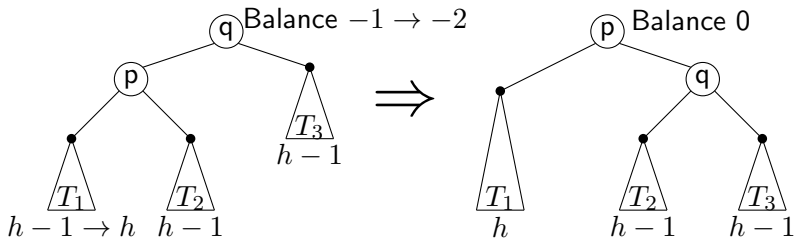


2. Fall:

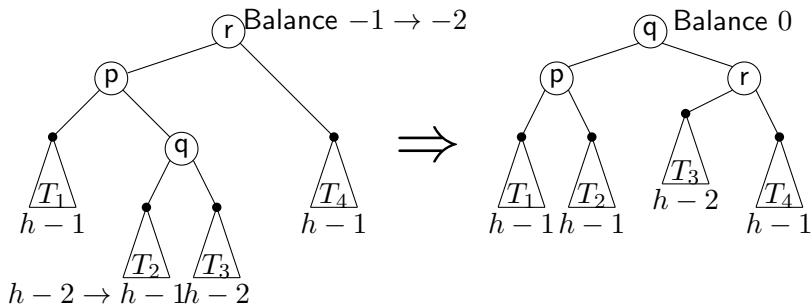


Hier ist eventuell eine Rebalancierung auf dem Pfad zur Wurzel notwendig, denn die Höhe des Unterbaums ändert sich.

Fall 2a:



Fall 2b:



- ③ *Delete*: Auch diese Operation wird wie bei natürlichen Suchbäumen implementiert. Jedoch hat am Ende ggf. noch eine Rebalancierung auf dem Pfad von dem Blatt, das an die Stelle des zu löschenden Elements geschrieben wird, zur Wurzel zu erfolgen.

Satz 22

Bei AVL-Bäumen sind die Operationen `IsElement`, `Insert`, und `Delete` so implementiert, dass sie die Zeitkomplexität $\mathcal{O}(\log n)$ haben, wobei n die Anzahl der Schlüssel ist.

Beweis:

Klar!



Im Grundsatz gelten folgende Bemerkungen für AVL-Bäume:

- 1 Sie haben in der Theorie sehr schöne Eigenschaften, auch zur Laufzeit.
- 2 Sie sind in der Praxis sehr aufwändig zu implementieren.

Weitere Informationen:



Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman:
The design and analysis of computer algorithms,
Addison-Wesley Publishing Company: Reading (MA), 1974



Robert Sedgewick, Philippe Flajolet:
An introduction to the analysis of algorithms,
Addison-Wesley Publishing Company, 1996

3.3 Gewichtsbalancierte Bäume

Siehe zu diesem Thema Seite 189ff in



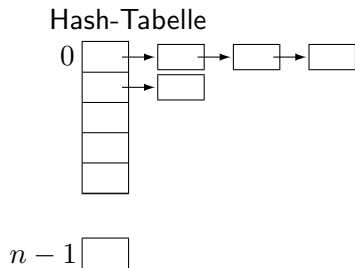
Kurt Mehlhorn:

Data structures and algorithms 1: Sorting and searching,
EATCS Monographs on Theoretical Computer Science,
Springer Verlag: Berlin-Heidelberg-New York-Tokyo, 1984

4. Hashing

4.1 Grundlagen

- Universum U von Schlüsseln, z.B. $\subseteq \mathbb{N}_0$, $|U|$ groß
- Schlüsselmenge $S \subseteq U$, $|S| = m \leq n$
- array $T[0..n-1]$ Hashtabelle
- Hashfunktion $h : U \rightarrow [0..n-1]$



Problemstellung: Gegeben sei eine Menge M von Elementen, von denen jedes durch einen Schlüssel k aus der Menge U bezeichnet sei. Die Problemstellung ist: Wie muss die Speicherung der Elemente aus M bzw. der zugehörigen Schlüssel organisiert werden, damit jedes Element anhand seines Schlüssels möglichst schnell lokalisiert werden kann?
Gesucht ist also eine Abbildung

$$h : U \rightarrow T$$

von der Menge aller Schlüssel in den Adressraum T der Maschine. Hierbei soll jedoch eine bisher nicht beachtete Schwierigkeit berücksichtigt werden: Die Menge U der möglichen Schlüsselwerte ist wesentlich größer als der Adressraum. Folglich kann die Abbildung h nicht injektiv sein, es gibt Schlüssel k_1, k_2, \dots mit $h(k_1) = h(k_2) = \dots$

Wir werden sehen, dass aufgrund dieses Umstandes die Speicherstelle eines Elements mit Schlüssel k von einem anderen Element mit einem anderen Schlüssel l besetzt sein kann und mit einer gewissen Wahrscheinlichkeit auch sein wird: Es treten **Kollisionen** auf.

Satz 23

In einer Hashtabelle der Größe n mit m Objekten tritt mit Wahrscheinlichkeit

$$\geq 1 - e^{-\frac{m(m-1)}{2n}} \approx 1 - e^{-\frac{m^2}{2n}}$$

mindestens eine Kollision auf, wenn für jeden Schlüssel jede Hashposition gleich wahrscheinlich ist.

Beweis:

Sei A_m das Ereignis, dass unter m Schlüsseln **keine** Kollision auftritt. Dann gilt

$$\begin{aligned}\Pr[A_m] &= \prod_{j=0}^{m-1} \frac{n-j}{n} = \prod_{j=0}^{m-1} \left(1 - \frac{j}{n}\right) \\ &\leq \prod_{j=0}^{m-1} e^{-\frac{j}{n}} = e^{-\sum_{j=0}^{m-1} \frac{j}{n}} = e^{-\frac{m(m-1)}{2n}}.\end{aligned}$$

Es folgt die Behauptung. □

Korollar 24

Hat eine Hashtabelle der Größe n mindestens $\omega(\sqrt{n})$ Einträge und ist für jeden Schlüssel jede Hashposition gleich wahrscheinlich, so tritt mit Wahrscheinlichkeit $1 - o(1)$ mindestens eine Kollision auf.

Um die Kollisionszahl möglichst gering zu halten, müssen Hashfunktionen gut streuen.

Definition 25

- 1 Eine Hashfunktion

$$h : U \rightarrow \{0, 1, \dots, n - 1\}$$

heißt **perfekt** für $S \subseteq U$, wenn für alle $j, k \in S, j \neq k$ gilt

$$h(j) \neq h(k) .$$

- 2 Eine Klasse \mathcal{H} von Hashfunktionen $h : U \rightarrow \{0, 1, \dots, n - 1\}$ heißt **perfekt**, falls \mathcal{H} für jedes $S \subseteq U$ mit $|S| = n$ eine für S perfekte Hashfunktion enthält.

Grundsätzliche Fragestellungen:

- 1 Wie schwierig ist es, perfekte Hashfunktionen darzustellen (also: was ist ihre Programmgröße)?
- 2 Wie schwierig ist es, gegeben S , eine für S perfekte Hashfunktion zu finden?
- 3 Wie schwierig ist es, gegeben $k \in S$, $h(k)$ für eine für S perfekte Hashfunktion auszuwerten?

Typische „praktische“ Hashfunktionen:

$$h(k) = k \bmod n \quad (\text{Teilermethode})$$

$$h(k) = \lfloor n(ak - \lfloor ak \rfloor) \rfloor \quad \text{für } a < 1 \quad (\text{Multiplikationsmethode})$$

Wir betrachten zunächst Methoden der Kollisionsbehandlung.

4.2 Methoden zur Kollisionsauflösung

Wir unterscheiden grundsätzlich

- geschlossene und
- offene Hashverfahren.

Bei geschlossenen Hashverfahren werden Kollisionen nicht wirklich aufgelöst.

4.2.1 Geschlossene Hashverfahren (Chaining)

Die Hashtabelle ist ein Array von n linearen Listen, wobei die i -te Liste alle Schlüssel k beinhaltet, für die gilt:

$$h(k) = i.$$

Zugriff: Berechne $h(k)$ und durchsuche die Liste $T[h(k)]$.

Einfügen: Setze neues Element an den Anfang der Liste.

Sei

$$\delta_h(k_1, k_2) = \begin{cases} 1 & \text{falls } h(k_1) = h(k_2) \text{ und } k_1 \neq k_2 \\ 0 & \text{sonst} \end{cases}$$

und

$$\delta_h(k, S) = \sum_{j \in S} \delta_h(j, k), \text{ Anzahl Kollisionen von } k \text{ mit } S.$$

Die Zugriffskosten sind:

$$\mathcal{O}(1 + \delta_h(k, S))$$

Sei A eine Strategie zur Kollisionsauflösung. Wir bezeichnen im Folgenden mit

- A^+ den mittleren Zeitbedarf für eine **erfolgreiche** Suche unter Verwendung von A ;
- A^- den mittleren Zeitbedarf für eine **erfolglose** Suche unter Verwendung von A ;
- $\alpha := \frac{m}{n}$ den **Füllfaktor** der Hashtabelle.

Sondierungskomplexität für Chaining

Falls auf alle Elemente in der Hashtabelle mit gleicher Wahrscheinlichkeit zugegriffen wird, ergibt sich

- A^- : mittlere Länge der n Listen; da $\frac{m}{n} = \alpha$, folgt

$$A^- \leq 1 + \alpha.$$

- A^+ :

$$\begin{aligned} A^+ &= \frac{1}{m} \sum_{i=1}^m \left(1 + \frac{i-1}{n} \right) \\ &= \frac{1}{m} \sum_{i=0}^{m-1} \left(1 + \frac{i}{n} \right) \\ &= 1 + \frac{m(m-1)}{2nm} \\ &\leq 1 + \frac{m}{2n} = 1 + \frac{\alpha}{2} \end{aligned}$$

Für festen Füllfaktor α ergibt sich also im Mittel Laufzeit $\Theta(1)$.