

(2,4)-Bäume und Rot-Schwarz-Bäume

Florian Seitz

28. Juni 2001

Inhaltsverzeichnis

1	Sinn und Zweck	2
2	(2,4)-Bäume	2
2.1	Definition und Eigenschaften	2
2.2	Einfügen	2
2.3	Löschen	3
2.4	Beweis der Höhe	4
2.4.1	minimale Anzahl von Schlüsseln	4
2.4.2	maximale Anzahl von Schlüsseln	5
2.5	Komplexität	5
3	Rot-Schwarz-Bäume	6
3.1	Definition und Eigenschaften	6
3.1.1	Die 'schwarze'-Höhe	6
3.2	Einfügen	6
3.3	Löschen	7
3.4	Beweis der Höhe	7
3.5	Komplexität	8
4	Literaturverzeichnis	9

1 Sinn und Zweck

Die Motivation dazu finden wir im Bereich der **Datenbanken**. Dort werden Datenmengen schnell so gross, dass man sie nicht mehr im Hauptspeicher halten kann. Die Daten müssen auf einen Hintergrundspeicher, z.B. eine Festplatte, geschrieben werden. Ein Zugriff ist meist recht zeitintensiv. Darüber hinaus ist es ineffizient, einzelne Bytes aus einem Sekundärspeicher zu lesen. Auf solchen Speichern sind die Daten meist in grossen Blöcken organisiert. Bei einer Anfrage wird gleich der gesamte Block in den Hauptspeicher bzw. Puffer kopiert, dort ausgewertet und zurückgeschrieben. Die Referenzen auf diese Daten sind im Gegensatz zu den Datenmengen sehr klein und können problemlos im Hauptspeicher gehalten werden. Diese Referenzen können z.B. in einem (2,4)-Baum oder Rot-Schwarz-Baum organisiert werden.

2 (2,4)-Bäume

2.1 Definition und Eigenschaften

1. Bei einem (2,4)-Baum ist jeder Weg von der Wurzel zu einem Blatt gleich lang.
2. Außerdem besitzt jeder Knoten minimal zwei Einträge und maximal vier Einträge. Außer die Wurzel, die hat minimal einen Eintrag.
3. Die Einträge in einem Knoten werden sortiert gehalten.
4. Innere Knoten mit n Einträgen haben immer $n+1$ Kinder.
5. Seien S_1, \dots, S_n die Schlüssel eines Knoten und V_0, V_1, \dots, V_n seien die Verweise auf diese Kinder, dann gilt:
 - V_0 weist auf den Teilbaum kleiner als S_1
 - $V_i (i = 1, \dots, n - 1)$ weist auf den Teilbaum, dessen Schlüssel zwischen S_i und S_{i+1} liegen
 - V_n weist auf den Teilbaum mit Schlüsseln größer als S_n
 - In den Blattknoten sind die Zeiger nicht definiert.

2.2 Einfügen

Beim Einfügen muss zuerst das Blatt, in das der Schlüssel eingefügt werden soll, gesucht werden. Anschließend sind folgende Fälle zu berücksichtigen:

1. Einfügen in den leeren Baum:
Ein Wurzelknoten wird erzeugt und der Schlüssel eingefügt.
2. Einfügen in einen schon vorhandenen Knoten **ohne** Überlauf:
d.h., dass maximal drei Schlüssel im Knoten vorhanden sind und der vierte hinzugefügt werden soll. Der Schlüssel wird sortiert in den Knoten eingefügt.
3. Einfügen in einen schon vorhandenen Knoten **mit** Überlauf:
d.h., dass schon vier Schlüssel im Knoten vorhanden sind und ein fünfter hinzugefügt werden soll. Dann muss der Knoten aufgeteilt werden. Der Knoten v wird durch zwei neue Knoten v' und v'' ersetzt. v' bekommt die ersten beiden Schlüssel von v und v'' bekommt die letzten beiden Schlüssel von v . Das mittlere Element wird zum Vaterknoten geschickt und dort eingefügt. Falls der Vaterknoten auch überläuft, so wird das ganze rekursiv behandelt und der Überlauf läuft den Baum hoch.

2.3 Löschen

Zuerst muss der zu löschende Schlüssel gesucht werden. Wenn die Suche nicht erfolgreich war, so kann man z.B. eine Fehlermeldung ausgeben. Wenn die Suche erfolgreich war, so müssen folgende Fälle beachtet werden:

1. Löschen in einem Blatt

- (a) mit minimal drei Schlüsseln:
Da die Kinder-Zeiger in einem Blatt nicht definiert sind und das Blatt nach dem Löschvorgang **nicht** zu wenige Einträge enthält, kann man den Schlüssel einfach aus dem Blatt löschen, ohne etwas zu beachten.
- (b) mit maximal zwei Einträgen (das Blatt ist minimal ausgelastet):
D.h., dass nach dem Löschen das Blatt nur noch einen Schlüssel enthält. Das wird als 'Unterlauf' bezeichnet. Nach den Eigenschaften eines (2,4)-Baumes darf das nicht sein und somit muss ein Ausgleich geschaffen werden, der folgendermaßen aussieht:
Wenn der rechte Blatt-Nachbarn nicht minimal ausgelastet ist, so wird der linke Schlüssel des Nachbarn in den Vater geschrieben und das Element im Vater wird an die Stelle des gelöschten Eintrages geschrieben. Analog sieht es aus, wenn der linke Blatt-Nachbar nicht minimal ausgelastet ist.
Wenn die Blatt-Nachbarn alle minimal ausgelastet sind, aber der

Vater nicht, so wird wie folgt verfahren:

Das Blatt, in dem gelöscht wurde, und ein Nachbar werden zusammengefügt. Das wird als 'merge' bezeichnet. Der Schlüssel im Vater, der die beiden Blätter als Kinder hat, wird ebenfalls in das Blatt eingefügt. Somit hat der Vater einen Schlüssel weniger. Wenn jetzt der Vater nur noch einen Eintrag enthält, so wird rekursiv verfahren und er Unterlauf läuft den Baum hoch.

2. Löschen in einem inneren Knoten:

Das nächstgrößere Element (oder nächstkleinere Element, je nach Art der Implementierung) wird an die gelöschte Stelle kopiert und in dem Blatt gelöscht. Das nächstgrößere (bzw. nächstkleinere) Element findet man, in dem man im rechten Teilbaum, den linkesten Schlüssel sucht (bzw. im linken Teilbaum, den rechtesten Schlüssel sucht).

2.4 Beweis der Höhe

Es werden die Anzahl der Schlüssel mit k und die Höhe mit h bezeichnet.

2.4.1 minimale Anzahl von Schlüsseln

Minimale Anzahl von Schlüsseln heißt, dass jeder Knoten minimal ausgelastet ist. Das bedeutet wiederum, dass der Baum die **maximale** Höhe hat. Die Anzahl der Schlüssel sieht wie folgt aus:

Die Wurzel hat genau ein Element. Die zwei Kinder der Wurzel haben genau k , also genau zwei Einträge. Folglich gibt es $k + 1$ Kinderknoten, die wiederum k Elemente enthalten. Diese haben wieder $k + 1$ Kinder, usw.

In einer Tabelle zusammengefasst sieht das so aus:

Einträge je Ebene	Knoten je Ebene	Höhe
1	1	0
$2k$	2	1
$2k * (k + 1)$	$2(k + 1)$	2
$2k * (k + 1)^{i-1}$	$2(k + 1)^{i-1}$	i

Daraus ergibt sich für die Anzahl der inneren Schlüssel:

$$n = 1 + \sum_{i=1}^h 2k * (k + 1)^{i-1}$$

Wenn man diese Gleichung auflöst, erhält man für die Höhe:

$$h \leq \log_{k+1}\left(\frac{n+1}{2}\right)$$

2.4.2 maximale Anzahl von Schlüsseln

Die maximale Anzahl heißt, dass jeder Knoten maximal ausgelastet ist. Das bedeutet wiederum, dass der Baum die **minimale** Höhe hat. Die Anzahl der Schlüssel sieht wie folgt aus:

Das läßt sich analog zur minimalen Anzahl von Schlüsseln berechnen. Die Wurzel hat allerdings $2k$ Einträge und daraus folgt, dass es genau $2k + 1$ Kinder gibt. Diese Kinder haben wiederum $2k$ Einträge und jedes Kind hat $2k + 1$ Kinder. Die Anzahl der Schlüssel der Wurzelkinder beläuft sich auf $2k * (2k + 1)$. Wiederum tabellarisch zusammengefasst:

Einträge je Ebene	Knoten je Ebene	Höhe
$2k$	1	0
$2k * (2k + 1)$	$2k + 1$	1
$2k * (2k + 1)^2$	$(2k + 1)^2$	2
$2k * (2k + 1)^i$	$(2k + 1)^i$	i

Für die Anzahl der inneren Schlüssel ergibt sich folgende Gleichung:

$$n = \sum_{i=0}^h 2k * (2k + 1)^i$$

Wenn man diese Gleichung auflöst, erhält man für die Höhe:

$$h \geq \log_{2k+1}(n + 1) - 1$$

2.5 Komplexität

Nachfolgend werden definiert:

h := Höhe; n := Anzahl der inneren Schlüssel; k := Anzahl der Einträge im Knoten

Wie vorher bewiesen ergibt sich für die Höhe:

$$\log_{2k+1}(n + 1) - 1 \leq h \leq \log_{k+1}\left(\frac{n+1}{2}\right)$$

Da die Höhe logarithmisch ist, wird das **Suchen** $O(\log n)$ Zeit in Anspruch nehmen. Das ist wie bei einem Binärbaum.

Das **Einfügen** besteht im wesentlichen nur aus dem Suchen und dem Aufteilen, wenn ein Überlauf vorliegt. Das Aufteilen kann in $O(1)$ realisiert werden, da nur die Zeiger umgesetzt werden müssen und ein neuer Knoten erzeugt werden muss. Allerdings kann das Aufteilen bis zur Wurzel hochlaufen und somit ergibt sich für das Einfügen auch die Komplexität von $O(\log n)$.

Beim **Löschen** muss ebenso wie beim Einfügen zuerst nach dem Schlüssel gesucht werden. Dann wenn es beim Löschen zu einem Unterlauf kommt,

muss der 'Merge'-Schritt gemacht werden. Dieser kann ebenfalls in $O(1)$ implementiert werden. Beim 'Mergen' müssen nur die Zeiger umgesetzt werden und ein Knoten gelöscht werden. Also ergibt sich auch beim Löschen eine Komplexität von $O(\log n)$.

3 Rot-Schwarz-Bäume

3.1 Definition und Eigenschaften

1. Jeder Knoten ist rot oder schwarz.
2. Jeder innere Knoten hat genau zwei Kinder.
3. Jedes Blatt (null) ist schwarz.
4. Kinder von einem roten Knoten sind schwarz.
5. Auf jedem Pfad von der Wurzel zu einem Blatt ist die Anzahl der schwarzen Knoten gleich.
6. Jeder innere Knoten speichert die 'schwarze'-Höhe und jedes Blatt hat die Höhe 0.

3.1.1 Die 'schwarze'-Höhe

Die 'schwarze'-Höhe wird von unten nach oben berechnet. Jedes Blatt hat die Höhe 0. Jeder Vater eines Blattes, egal ob rot oder schwarz, hat die Höhe 1. Es sind folgende Übergänge möglich:

1. roter Knoten \rightarrow schwarzer Vater: Der Knoten und der Vater haben die gleiche Höhe.
2. roter Knoten \rightarrow roter Vater: ist nach Definition nicht erlaubt.
3. schwarzer Knoten \rightarrow Vater egal: Die Höhe des Vaters ist um eins größer als die des Knotens.

3.2 Einfügen

Das Einfügen bei einem Rot-Schwarz-Baum ist dem Einfügen in einen Binären Suchbaum sehr ähnlich. Zuerst muss man die richtige Einfügestelle suchen. Wenn der Schlüssel noch nicht vorhanden ist, so ist die Einfügestelle fest. Wenn der Schlüssel schon vorhanden ist, ist es jedem selbst überlassen, ob

der neue Schlüssel als linkes oder rechtes Kind des schon vorhandenen Knoten angehängt wird. Das ist dann Sache der Implementierung. Nachdem die richtige Stelle gefunden und der Wert eingefügt wurde, wird der neue Knoten rot gefärbt und zwei neue schwarze Blätter angehängt. Es sind nun zwei Fälle zu unterscheiden:

- Wenn der Vater und der Onkel rot sind, so wird die Ordnung durch Umfärben wiederhergestellt. Eventuell muss bis zur Wurzel umgefärbt werden.
- Wenn der Vater rot und der Onkel schwarz ist, so stellt man die Ordnung durch Umfärben und Rotation wieder her. Dies kann maximal auch bis zur Wurzel gehen.

3.3 Löschen

Zuerst muss der zu löschende Schlüssel gesucht werden. Im folgenden wird vorausgesetzt, dass der entsprechende Knoten gefunden wurde. Bei zwei gleichen Schlüsseln, wird immer der 'unterste' Knoten genommen. Es können zwei Fälle auftreten:

- Wenn der zu löschende Knoten rot ist, so wird er gelöscht, ohne dass umgefärbt oder rotiert werden muss.
- Wenn der zu löschende Knoten schwarz ist, so wird die Ordnung durch Umfärben und/oder Rotation wieder hergestellt.

Zum zweiten Fall:

Es wird 'nur' umgefärbt, wenn der Nachbar vom gelöschten Knoten keine Kinder hat. Also wird der Vater und der Nachbar jeweils umgefärbt.

Umgefärbt und rotiert wird in allen anderen Fällen. Die Kante zum gelöschten Knoten wird dann markiert (als *double black* bezeichnet) und der Vater umgefärbt. Da der Nachbar vom gelöschten Knoten Kinder hat, wird rotiert. Wenn der Nachbar schwarz ist, so sind keine weiteren Schritte nötig, um die Ordnung wieder herzustellen. Allerdings, wenn der Nachbar rot ist, so müssen weitere Umfärbungen vorgenommen werden, um die 'schwarze'-Höhe wieder herzustellen.

3.4 Beweis der Höhe

Es werden die Anzahl innerer Schlüssel mit n und die 'schwarze'-Höhe mit h bezeichnet. Mit $bh(x)$ ist die 'schwarze'-Höhe des Knoten x gemeint.

Als Annahme halten wir die Anzahl innerer Knoten des Unterbaumes x fest

als: $x \geq 2^{bh(x)} - 1$. Dies wird mit Induktion über die Höhe von x bewiesen.

Sei $x = 0$: x ist dann ein Blatt(null) und auf der rechten Seite gilt: $2^{bh(x)} - 1 = 2^0 - 1 = 0$ innere Knoten, also genau ein Blatt.

Beim Übergang von $x \mapsto x + 1$ betrachten wir einen inneren Knoten mit positiver Höhe und zwei Kindern. Die 'schwarze'-Höhe jedes Kindes ist dann $bh(x)$ oder $bh(x) - 1$, abhängig von der Farbe des Kind-Knoten selbst.

\Rightarrow Die Höhe eines Kindes von x ist somit kleiner gleich als die Höhe von x selbst. Mit der Induktionsannahme folgt:

die Anzahl innerer Knoten jedes Kindes $\geq 2^{bh(x)-1} - 1$.

\Rightarrow Für die Anzahl innere Knoten vom Unterbaum x gilt nun:

$x \geq (2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$ q.e.d.

Wenn ein Knoten rot ist, so sind nach der Definition die Kinder schwarz. \Rightarrow Höchstens die Hälfte aller Knoten auf einem Weg von der Wurzel zu einem Blatt, außer der Wurzel, müssen somit schwarz sein. \Rightarrow Die 'schwarze'-Höhe von der Wurzel ist kleiner gleich $\frac{h}{2} \Rightarrow n \geq 2^{\frac{h}{2}} - 1$. Die Ungleichung nach h aufgelöst: $h \leq 2 \log_2(n + 1)$

Wenn nun alle Knoten auf dem Weg von der Wurzel zu einem Blatt schwarz sind, so ist die 'schwarze'-Höhe von der Wurzel $\leq h \Rightarrow n \leq 2^h - 1$. Wenn man die Ungleichung nach h auflöst, ergibt sich: $h \geq \log_2(n + 1)$ q.e.d.

Für die Höhe gilt nun:

$$\Rightarrow \log_2(n + 1) \leq h \leq 2 \log_2(n + 1)$$

3.5 Komplexität

Nachfolgend werden definiert:

h := Höhe; n := Anzahl der Knoten

Wie vorher bewiesen ergibt sich für die Höhe:

$$\log_2(n + 1) \leq h \leq 2 \log_2(n + 1)$$

Da die Höhe logarithmisch ist, wird das **Suchen** $O(\log n)$ Zeit in Anspruch nehmen. Das ist wie bei einem Binärbaum bzw. wie beim (2,4)-Baum.

Beim **Einfügen** muss zuerst die richtige Einfügestelle gesucht werden, also $O(\log n)$. Dann wird der Schlüssel eingefügt $O(1)$ und es muss eventuell noch umgefärbt bzw. rotiert werden. Das Umfärben bzw. Rotieren geht in $O(1)$, da hier nur Zeiger umgesetzt werden müssen. Es kann allerdings der Fall auftreten, dass bis zur Wurzel umgefärbt bzw. rotiert werden muss, dann nimmt das $O(1) * h = O(\log n)$ Zeit in Anspruch. Zusammenfassend kann man sagen, dass das Einfügen in $O(\log n)$ geht.

Beim **Löschen** ist es ebenso wie beim Einfügen. Zuerst den zu löschenden Knoten suchen, wenn er nicht gefunden wird, ist das Löschen vorbei

und man kann z.B. eine Fehlermeldung ausgeben. Wenn der Knoten gefunden worden ist, dann wird er gelöscht und eventuell noch Umfärbungs- bzw. Rotationsschritte vorgenommen. Umfärben und rotieren geht auch wieder in $O(1)$, da wieder nur Zeiger umgesetzt werden. Wenn diese Schritte bis zur Wurzel gehen, gilt $O(1) * h = O(\log n)$. Abschließend gilt für das Löschen, dass es insgesamt $O(\log n)$ Zeit kostet.

4 Literaturverzeichnis

T.H.Cormen, C.E.Leiserson, R.L.Rivest:
Introduction to Algorithms,
The MIT Press, Cambridge, Mass. u. a., 1990

M.T.Goodrich, R.Tamassia:
Data Structures and Algorithms in Java,
John Wiley & Sons, Inc., 2001

Dr. Volker Heun
Grundlegende Algorithmen
Vieweg, Braunschweig, 2000

Prof. Alfons Kemper, Ph.D.
Vorlesungsskript von Praktischer Informatik I
Passau, 2000