

Ausarbeitung zum Proseminar
„Algorithms and Data Structures for Database Systems”

von
Ludwig Bachmaier

Thema:
B - Bäume

Vortrag am 29.01.2003

Moderation:
Jan Ehrlich

Inhaltsverzeichnis

1. Einleitung	3
2. Definition und Eigenschaften eines B-Baums	4
2.1. Exakter Aufbau eines Knotens im B-Baum	4
2.2. Eigenschaften eines B-Baums	5
2.3. Bestimmung der Höhe eines B-Baums	6
3. Grundlegende Operationen mit B-Bäumen	6
3.1. Suchen von Werten	7
3.2. Einfügen	8
3.3. Löschen	8
3.3.1. Löschen in einem Blatt	9
3.3.2. Löschen in einem inneren Knoten	11
4. Varianten des B-Baums	12
4.1. B*-Bäume	12
4.2. B+-Bäume	13
4.3. Präfix B+-Bäume	15
5. Mehrbenutzersysteme und B-Bäume	15
6. Abschließende Bewertung des B-Baums	16
7. Literaturverzeichnis	17

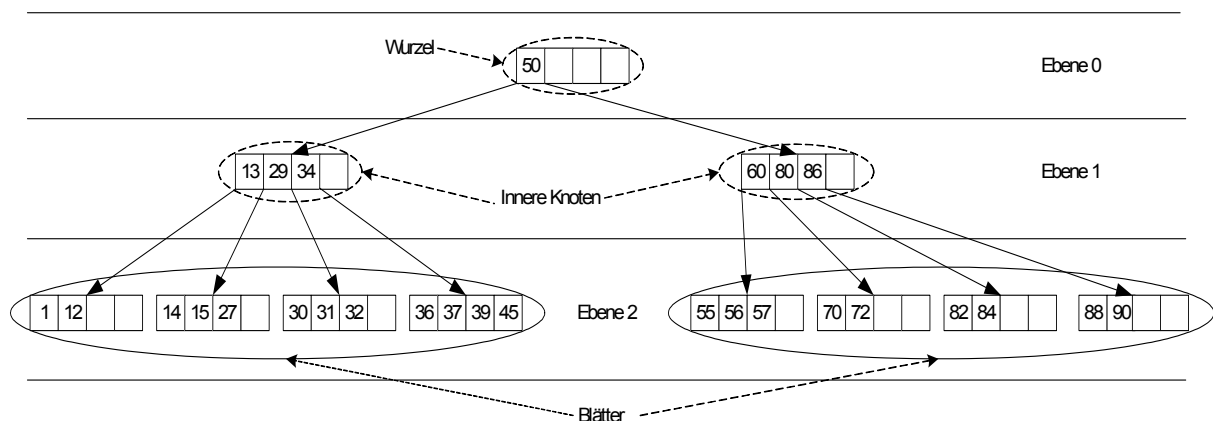
1. Einleitung

Um mit großen Datenmengen auf Massenspeichern (auf Bändern, Festplatten und dergleichen) effizient umgehen zu können, bedarf es raffinierter Techniken.

Bei Datenbanken etwa hat man das Problem, dass man nie alle Informationen im (Silizium-) Hauptspeicher behalten kann, entweder weil es zu teuer oder schlichtweg technisch unmöglich wäre - man stelle sich nur Daten im Umfang von einigen TeraByte vor, wie sie durchaus vorkommen. Um nun mit den langsamen (hohe Zugriffszeiten, niedrige Transferraten) aber unverzichtbaren (Magnet-)Hintergrundspeichern dennoch innerhalb vertretbarer Zeiten zu Resultaten zu gelangen, entwarfen Bayer und McCreight zu Beginn der 1970er Jahre die sog. B-Bäume. Diese Datenstruktur ist geradezu prädestiniert für den Einsatz auf langsamen Hintergrundspeichern. Neben dem ursprünglichen B-Baum gibt es noch einige Spielarten wie den B+-Baum und den B*-Baum, auf die später noch näher eingegangen wird.

Anhand von Abb. 1 ersieht man relativ schnell den Vorteil, den B-Bäume besitzen und warum sie in der Literatur auch unter dem Begriff „Vielwegebäume“ geführt werden (die gängigen Termini bei Baumstrukturen sind in Abb.1 nochmals eingezeichnet).

Im Gegensatz zu Binärbäumen haben B-Bäume mehrere Einträge in den Knoten und sie verzweigen sich auch nicht binär, sondern vielfach. Als Resultat erhält man so Strukturen, die eine sehr geringe Höhe aufweisen, damit mit einer geringen Anzahl von Plattenzugriffen auskommen und enorme Informationsmengen aufnehmen können.



(Abb. 1: Bsp. B-Baum vom Grad 2)

Näheres zur Organisation von Plattenspeichern findet man bei [Cormen], S.434ff; aus dieser Einleitung geht hervor, wie groß die Unterschiede bei den Zugriffszeiten zwischen Magnet- und Siliziumtechnik sind. Eine moderne Festplatte besitzt eine Zugriffszeit von etwa 3-9 Millisekunden (10^{-3} s), RAM-Speicher aus Silizium weist aber Zeiten auf, die sich im Nanosekundenbereich bewegen (10^{-9} s). Man erkennt, wie wichtig es ist, Zugriffe auf den externen Speicher zu minimieren, weil dieser um Größenordnungen langsamer ist.

2. Definition und Eigenschaften eines B-Baums

2.1. Exakter Aufbau eines Knotens im B-Baum

Es sollen nun einige Eigenschaften des B-Baums (auch anhand von Abb. 1) erläutert werden. Wie erwähnt, sind die Knoten im B-Baum komplexer als bei anderen Bäumen, das ergibt sich nicht nur aus der Anzahl von Einträgen, sondern auch aus dem generell erweiterten Aufbau eines Knotens.

Betrachtet man z.B. in Abb. 1 den linken Knoten auf der mittleren Ebene (Ebene 1) „vergrößert“, so erkennt man folgende Struktur (der rechte Knoten von Ebene 1 weist die gleiche Struktur auf):

V 0	Schlüssel 1	V 1	Schlüssel 2	V 2	Schlüssel 3	V 3	Im Beispiel leer!	V 4
	Datensatz 1		Datensatz 2		Datensatz 3			
								Hier leer!

(Abb. 2: Ausführliche Darstellung eines Knotens im B-Baum, vgl. [Kemper], S.207f)

Es gehen vier stark ausgezogene „Pfeile“ von den „vertikalen Linien“ in Wurzel und inneren Knoten (aus Abb. 1) weg.

Wie man jetzt aus Abb. 2 ersieht, sind diese vermeintlich einfachen „vertikalen Linien“ tatsächlich Verweise (auch Referenzen oder Pointer genannt) auf nachfolgende Knoten (die man auch Kinder, Nachfolger oder Söhne nennt).

Die Zahleneinträge in Abb. 1 entsprechen den Schlüsseln 1-3 in Abb. 2. Es fällt auf, dass die Schlüssel in den Knoten der Größe nach sortiert sind.

Im Beispiel sind nur drei Schlüssel/Datensatz-Felder sowie vier Verweise belegt.

Die Datensätze enthalten die eigentlichen Nutzdaten. Aus Gründen der Vereinfachung sollen die Datensätze aber auch zukünftig nicht näher betrachtet werden und auch die Verweise werden durch die einfachen „vertikalen Linien“ repräsentiert.

Die oben erwähnten „Pfeile“ die von den Verweisen wegführen (also den „Verweispfad“ verdeutlichen), besitzen die folgenden Eigenschaften:

V 0 zeigt auf einen Teil des Baums hin, dessen Schlüssel kleiner sind als die von Schlüssel 1 (im Beispiel: Schlüssel 1 [13] ist größer als 1 und 12).

V 1 weist auf einen Teilbaum hin, dessen Schlüssel zwischen Schlüssel 1 und Schlüssel 2 liegen (im Beispiel: 14, 15, 27 sind größer als Schlüssel 1 [13], aber kleiner als Schlüssel 2 [29]).

V 2 zeigt auf Schlüssel hin, die größer als der Schlüssel 2 [29], aber kleiner als Schlüssel 3 [34] sind.

V 3 ist der letzte Verweis; er zeigt auf ausschließlich größere Schlüssel hin (größer als 34).

V 4 und Datensatz/Schlüssel 4 sind nicht belegt.

Wichtig zu wissen ist, dass man die Größe der Knoten nicht beliebig wählt; dass man die Knoten eines B-Baums auch als „Seiten“ bezeichnet, ist schon ein Hinweis, wie die Knoten-/Seitengröße zu wählen ist:

Die Knoten werden abgebildet auf die Seiten des verwendeten Hintergrundspeichers, wobei die beiden Größen übereinstimmen sollen; man will mit einem Zugriff auf das externe Speichermedium genau eine Seite in den Hauptspeicher lesen. Klappt das nicht, weil die Seitengrößen nicht gleich sind, so werden erneute kostspielige Zugriffe auf das externe Medium notwendig.

Hintergrundspeicher verwenden üblicherweise eine Blockstruktur zur Speicherung von Daten (ein Block entspricht etwa 512 – 4096 Bytes); die Knotengröße eines B-Baums sollte der Blockgröße des verwendeten externen Speichers entsprechen.

Im Folgenden sollen die Begriffe „Seite“ und „Knoten“ jeweils synonym verwendet werden.

2.2. Eigenschaften eines B-Baums

Die anhand des Beispiels herausgearbeiteten Eigenschaften eines B-Baums sollen formaler und erweitert festgehalten werden:

Ein B-Baum vom Grad (auch Ordnung genannt) n hat folgende Eigenschaften:

- E1: Jeder Knoten enthält maximal $2n$ Elemente (Schlüssel). Die Elemente in den Knoten sind sortiert.
- E2: Jeder Knoten muss mindestens n Elemente enthalten; das gilt jedoch nicht für die Wurzel, sie darf auch weniger Elemente enthalten (siehe auch Abb. 1: nur ein Schlüssel in der Wurzel, obwohl eigentlich zwei erforderlich wären)
- E3: Die Knoten, die keine Nachfolger besitzen, nennt man Blätter; sie liegen alle auf einer Ebene; diese letzte Ebene bestimmt zugleich die Höhe des Baums. Blätter enthalten keine Verweise.
- E4: Die Wege von der Wurzel zu einem Blatt sind stets gleich lang.
- E5: Knoten, die keine Blätter sind, haben $m+1$ Nachfolger, wobei m die Anzahl der im jeweiligen Knoten gespeicherten Schlüssel ist (vgl. Abb.1 und 2: der betreffende Knoten hat $m = 3$ Schlüssel, aber $m + 1 = 4$ Verweise / Nachfolger).
- E6: „Ordnungskriterien“ im B-Baum (vgl. [Kemper], S. 208):
Seien S_1, \dots, S_n die Schlüssel eines Knotens der $m+1$ Nachfolger besitzt. V_0, \dots, V_n sollen die Verweise auf die genannten Nachfolger sein. Es gilt dann:
 - a) V_0 weist auf den Teilbaum mit Schlüsseln kleiner als S_1 hin.
 - b) V_i ($i = 1, \dots, n-1$) zeigt auf den Teilbaum hin, dessen Schlüssel zwischen S_i und S_{i+1} liegen.
 - c) V_n weist auf den Teilbaum hin, dessen Schlüssel größer sind als die von S_n .

Beispiele dazu wurden zu Beginn dieses Kapitels genannt.

B-Bäume sind stets vollständig ausgeglichen; alle Wege von der Wurzel zu den Blättern sind gleich lang.

Bedingt durch die zumeist hohe Anzahl von Verzweigungen, findet man gewünschte Elemente schnell und ohne viele Knoten durchwandern zu müssen, was gleichzeitig die Anzahl der Zugriffe auf den Hintergrundspeicher verringert. Man liest ja mit einem Zugriff eine Seite (zur Wiederholung: eine Seite entspricht einem Knoten!) in den internen Speicher ein, die vergleichsweise viel Information enthält.

Eine Seite ist gerade dann zu wechseln, wenn ein Element nicht darin gefunden wurde und man entlang einer weiteren Kante (eines „Pfeils“) zu einer neuen Seite übergehen muss.

Durch das Kriterium E2 wird sichergestellt, dass die Knoten zumindest zu 50 % belegt sind. Es existieren Varianten des B-Baums (B*-Bäume, mehr dazu später) die eine 2/3 Belegung der Knoten garantieren und somit Speicherplatz sparen.

2.3. Bestimmung der Höhe eines B-Baums

Die Höhe eines B-Baums ist wichtig, da sie unter anderem die Anzahl der Zugriffe auf ein Medium bestimmt (die Anzahl der Zugriffe ist proportional zur Höhe des Baums).

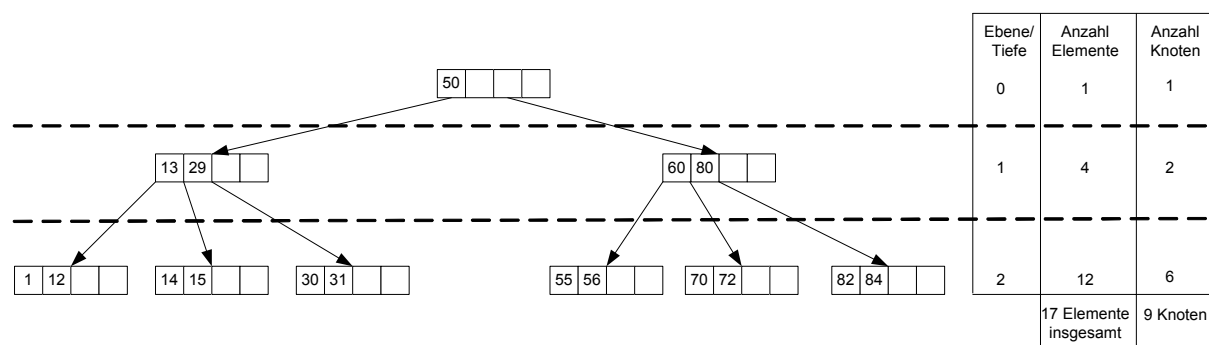
Zunächst berechnet man die minimale Anzahl von Einträgen im B-Baum, wobei man unter minimal versteht, dass der Baum vom Grad n genau einen Eintrag in der Wurzel hat, seine Knoten nur mit n Einträgen gefüllt sind und er somit eine minimale Anzahl von Knoten hat. Es ergibt sich folgender Ausdruck für die minimale Anzahl von Elementen ($\#$ sei stets die Anzahl von Elementen):

$$\#_{\text{Min}} = 1 + 2 \left(\sum_{i=0}^{h-1} (n+1)^i \right) n$$

Mittels Umformung erhält man schließlich

$$\#_{\text{Min}} = 2(n+1)^h - 1 \text{ als untere Grenze.}$$

Ein Beispiel soll die Minimalbelegung eines B-Baums demonstrieren:



(Abb. 3: Minimal belegter B-Baum vom Grad $n = 2$ und Höhe $h = 2$)

Die maximale Anzahl von Elementen (alle Knoten mit $2n$ Elementen gefüllt, somit maximale Anzahl von Nachfolgern) ergibt sich zu:

$$\#_{\text{Max}} = \left(\sum_{i=0}^h (2n+1)^i \right) 2n$$

Schließlich erhält man nach Umformung als obere Grenze:

$$\#_{\text{Max}} = (2n+1)^{h+1} - 1$$

Würde man das Beispiel von Abb.3 im Maximalfall weiterführen, so hätte man bereits einen Baum mit 124 Einträgen; bei Grad $n = 100$, $h = 2$ (n liegt in der Praxis zwischen 50 und 2000), ergeben sich 8.120.600 Einträge!

Die Anzahl von Elementen liegt somit im Bereich

$$2(n+1)^h - 1 \leq \# \leq (2n+1)^{h+1} - 1.$$

Die minimale Anzahl der Knoten ergibt sich zu $1 + \frac{2}{n} \left((n+1)^h - 1 \right)$.

Maximal hat ein B-Baum $\frac{1}{2n} \left((2n+1)^{h+1} - 1 \right)$ Knoten.

Man vgl. dazu [Bayer], S.174ff (die Höhe h wurde angepasst, entsprechend unserer Definition von h).

Die Höhe eines B-Baums vom Grad n bei einer gegebenen Anzahl von Einträgen # liegt nach obigen Erkenntnissen (Anzahl von Einträgen) innerhalb folgender Schranken:

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

Man sieht sofort, dass die Höhe und damit die Zugriffe auf den Hintergrundspeicher gering sein werden (die Basis des Logarithmus kann groß werden). Die Kosten einer Suchoperation sind lediglich logarithmisch und sie werden geringer, je größer die Basis des Logarithmus wird (damit verbunden ist natürlich die Anzahl der Verzweigungen). Analoges gilt auch für die Operationen zum Einfügen und Löschen wie sie unter 3. beschrieben werden.

3. Grundlegende Operationen mit B-Bäumen

3.1. Suchen von Elementen

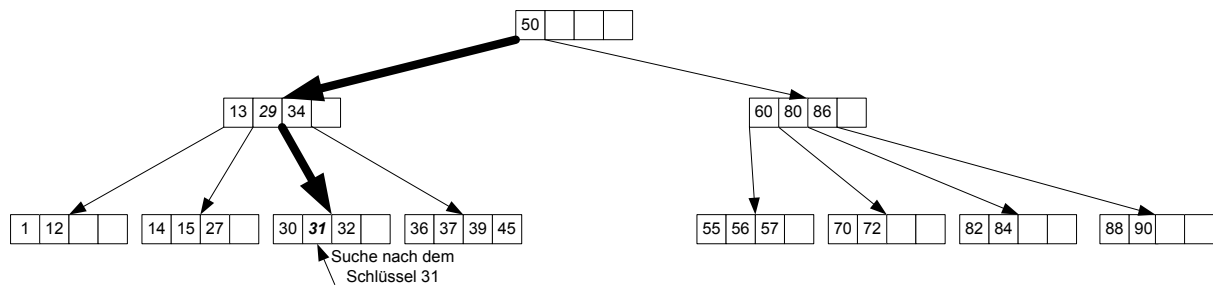
Zunächst soll das Suchen eines Schlüssels im Baum kurz erläutert werden.

Das Suchen in einem B-Baum ist dem Vorgehen in einem binären Suchbaum ähnlich, siehe dazu auch Eigenschaft E6. Man hat beim Binärbaum aber nur je zwei Auswahlmöglichkeiten, beim B-Baum je nach Grad entsprechend mehr; weiterhin muss man beim B-Baum auch die Knoten durchsuchen.

Anhand des Beispiels aus Abb. 4 lässt sich das Prinzip der Suche erkennen.

Will man die Zahl 31 suchen, so weiß man, dass sie im linken Unterbaum (auf den der linke Verweis des Schlüssels 50 zeigt, da $31 < 50$) zu finden sein wird. Man darf nicht vernachlässigen, dass eigentlich auch der Wurzelknoten Eintrag für Eintrag durchsucht werden müsste; hier erübrigt sich das, da nur ein Schlüssel vorhanden ist.

Den Knoten auf Ebene 1 durchsucht man Schlüssel für Schlüssel, den gewünschten Eintrag wird man dort nicht finden, allerdings muss man nach der 29 und noch vor der 34 eine Stufe hinab steigen (dem entsprechenden Pointer folgen). Auf Ebene 2 durchsucht man schließlich den entsprechenden Knoten von links nach rechts und wird beim zweiten Element fündig.



(Abb. 4: Beispiel zur Suche im B-Baum; Suche nach dem Schlüssel 31)

3.2. Einfügen von Schlüsseln

Eine andere Basisoperation ist das Einfügen von neuen Werten.

Man sucht mit Hilfe des in 3.1. genannten Verfahrens zunächst nach der passenden Position im Baum. Hat man die entsprechende Stelle im Knoten gefunden, so ergeben sich zwei Fälle:

1. Fall

Ist der Knoten noch nicht vollständig gefüllt (er hat weniger als $2n$ Einträge), so fügt man den neuen Schlüssel einfach ein, wobei die Größe der Schlüssel von links nach rechts zunimmt (siehe auch Eigenschaft E1 aus Kapitel 2).

2. Fall

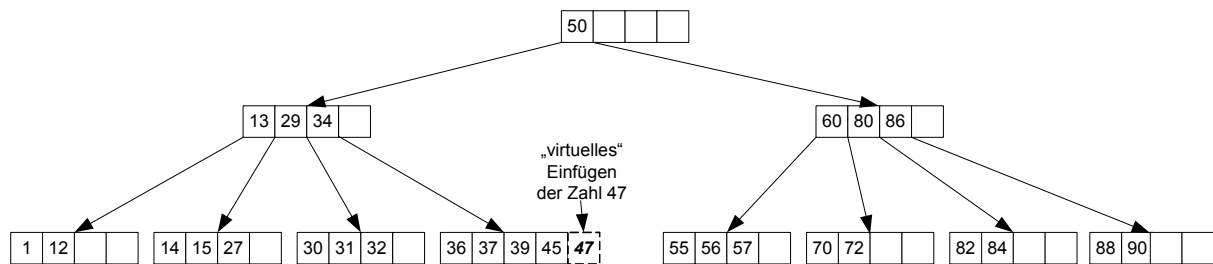
Kommt es hingegen im betreffenden Knoten zu einem „Überlauf“, d.h. er würde mehr als $2n$ Einträge enthalten, so fügt man den neuen Schlüssel erst einmal „virtuell“ ein, d.h. man erweitert den Knoten um einen neuen Eintrag, und fügt dann den neuen Schlüssel ein, wobei man die Größe der Schlüssel berücksichtigen muss. Anschließend bestimmt man den Eintrag in der Mitte des virtuellen Knotens. Der Knoten wird gespalten, wobei die Schlüssel, die links vom mittleren Schlüssel liegen in einen neuen linken Knoten eingetragen werden; die Einträge, die rechts vom „Mittelwert“ liegen, werden in einen neuen rechten Knoten eingetragen; aus einem Knoten wurden somit zwei, die beide wieder alle Eigenschaften eines B-Baums erfüllen und mit n Einträgen gefüllt sind.

Der vorgenannte mittlere Eintrag wird in den Vaterknoten (der Knoten, der sich eine Ebene über dem virtuellen Knoten befindet) verschoben. Sollte dieser eine Ebene höher gelegene Knoten nun aber ebenfalls voll sein, so wird auch er nach der genannten Regel geteilt. Dieses Verfahren wird nun unter Umständen solange durchgeführt bis man an der Wurzel angelangt ist.

Ein Beispiel soll diese Prozedur verdeutlichen:

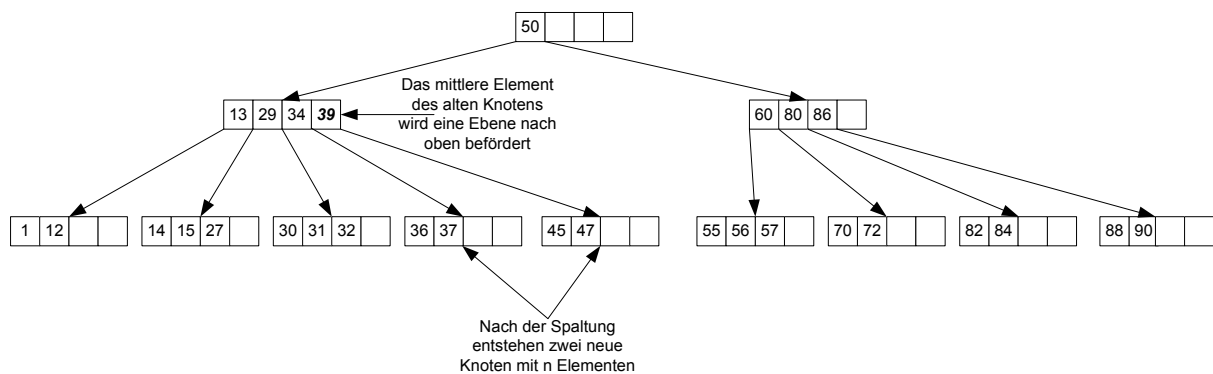
Das Element 47 soll in den Baum unten (Abb. 5) eingefügt werden; nachdem man den passenden Knoten gefunden hat, stellt man jedoch fest, dass er bereits mit der maximalen Anzahl von Elementen belegt ist (bei Grad $n = 2$ also vier Elemente, siehe Eigenschaft E1).

Man erweitert nun den Knoten auf fünf Einträge, bestimmt das mittlere Element (hier die 39), teilt den Knoten wie erläutert und verschiebt die 39 in den Vaterknoten, wie in Abb. 5 gezeigt wird.



(Abb. 5: Einfügen der Zahl 47)

Um den Anforderungen die an einen B-Baum gestellt werden gerecht zu werden, muss der „virtuelle“ Knoten wie erläutert geteilt werden. Im vorliegenden Fall ist die 39 das mittlere Element, so dass diese Zahl eine Ebene nach oben gereicht wird und folgender Baum entsteht:



(Abb. 6: Spalten des Knotens nach einer Einfügeoperation und Verschieben des mittleren Elements)

Die Höhe eines B-Baums ändert sich dann, wenn sich die Spaltungsoperationen bis zur Wurzel hin fortsetzen.

3.3. Löschen von Schlüsseln

Die nächste Operation die in einem Baum durchgeführt werden kann, ist das Löschen von Elementen.

3.3.1. Löschen in einem Blatt

Liegt der zu entfernende Schlüssel in einem Blatt, so wird der entsprechende Schlüssel gelöscht, Elemente die rechts vom entfernten Schlüssel liegen werden nach links verschoben. Sind nach dem Löschvorgang weniger als n Schlüssel (Unterlauf, der die Eigenschaft E2 verletzt!) im betreffenden Blatt, so unterscheidet man zwei Fälle:

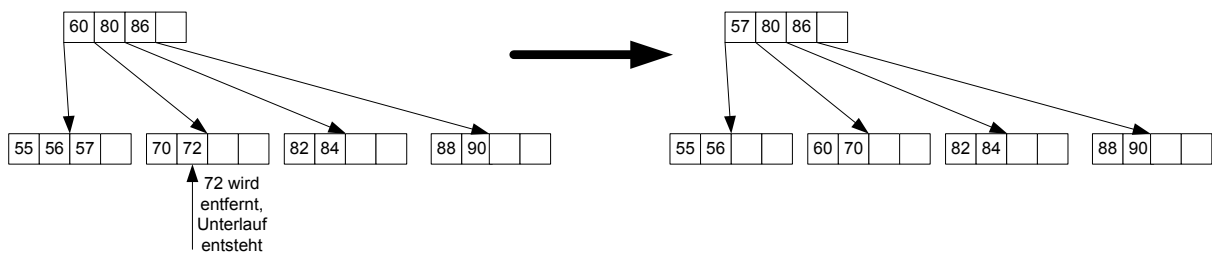
1. Fall:

Kann der Nachbarknoten mindestens ein Element abgeben (weil er mehr als n Einträge besitzt), so werden die Einträge zwischen den beiden Knoten so aufgeteilt, dass beide Knoten die gleiche Anzahl von Elementen besitzen (die Ordnungskriterien müssen beachtet werden!).

Unter Nachbarknoten versteht man einen Knoten der denselben Vater hat; außerdem müssen die Verweise, die vom Vaterknoten auf die Nachfolger zeigen, benachbart sein – in Abb.7 wären die jeweils ganz außen gelegenen linken und rechten Knoten keine Nachbarn, da die Verweise, die auf sie hindeuten nicht benachbart sind.

Um die Ordnung im Baum aufrecht zu erhalten, müssen auch am Vaterknoten Änderungen vorgenommen werden, die einer Rotation ähneln. Hat der linke Nachbar (kleinere Elemente, wie in Abb. 7) Einträge, die er abgeben kann, so wird das größte Element des Knotens (das am weitesten rechts steht) in den Vaterknoten verschoben, wo es die Position des Eintrags übernimmt, von dem Verweise auf die am Ausgleich beteiligten Knoten hinzeigen. Der „von seinem Platz verdrängte“ Eintrag im Vaterknoten wird in den Knoten „gedrückt“, in dem der Unterlauf besteht; er nimmt dort den Platz des kleinsten Eintrags ein.

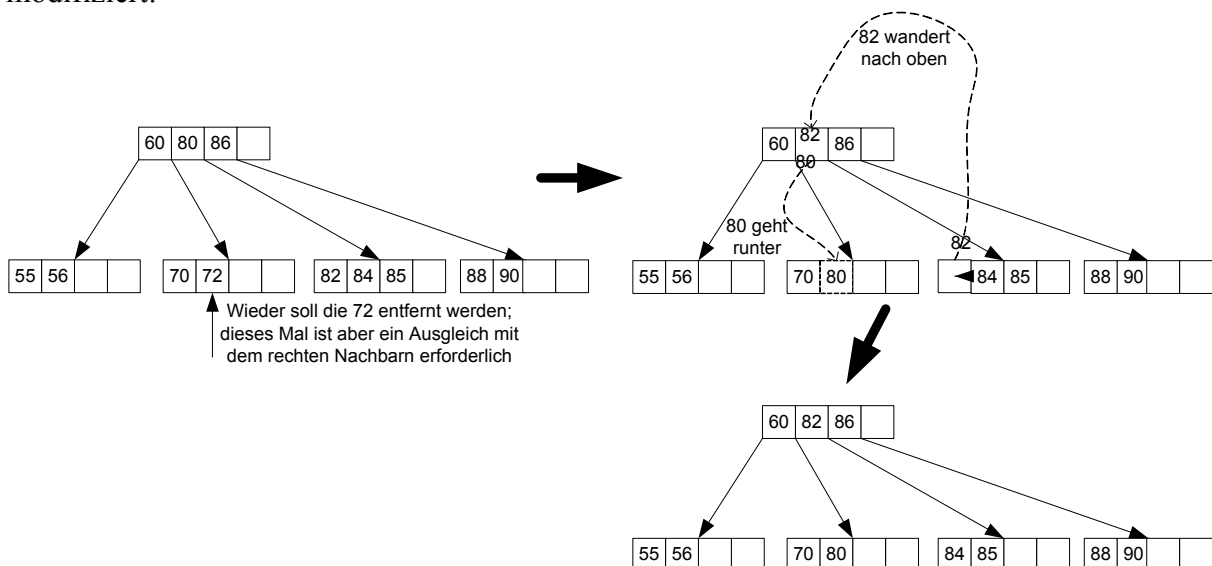
Ein Beispiel (anhand des rechten Teilbaums der obigen Demonstrationen) dazu:



(Abb. 7: Löschen des Schlüssels 72)

Man erkennt, dass eine Art von Rotation durchgeführt wurde; der Eintrag mit Schlüssel 57 wurde in den Vaterknoten geschoben, der Schlüssel 60 wurde zum Ausgleich in den Knoten „gedrückt“, der einen Unterlauf produzierte.

Analog dazu kann es folgenden zweiten Fall geben, das obige Beispiel wurde dazu leicht modifiziert:



(Abb. 8: wieder Löschen der 72, aber andere Rotationsrichtung)

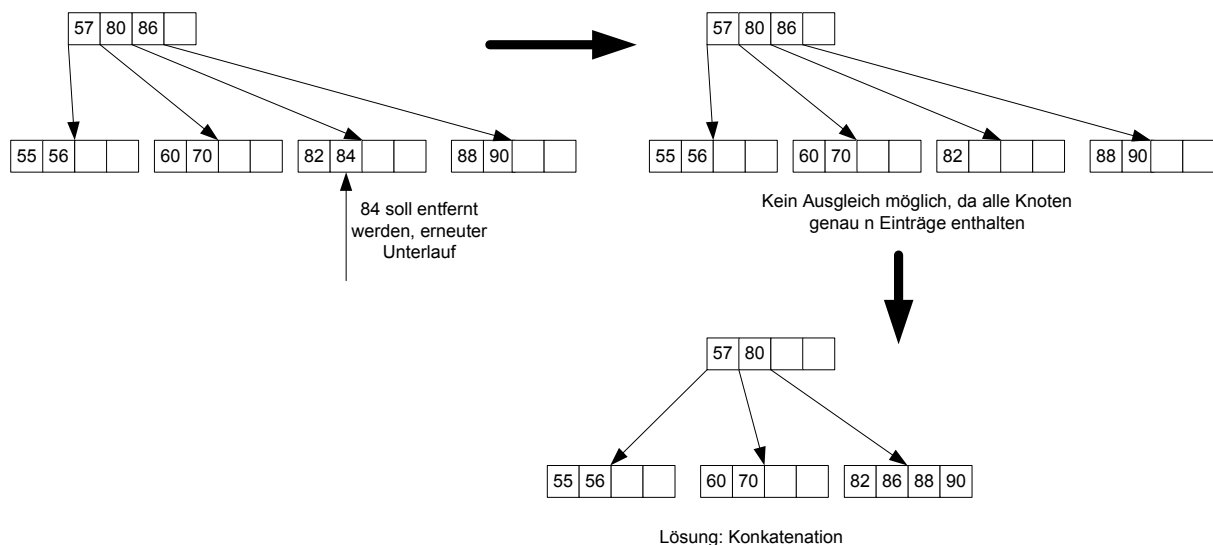
In Abb. 8 muss der rechte Nachbar zum Ausgleich verwendet werden. Dazu wird das kleinste Element dieses Nachbars in den Vaterknoten geschoben, wo es den Eintrag verdrängt, dessen Verweise auf die beiden am Ausgleichsverfahren beteiligten Knoten zeigen. Im vorliegenden

Fall ist das der Schlüssel 80, der dann sogleich in den linken Nachfolger hinuntergedrückt wird, wo er die Position des größten Elements einnimmt.

2. Fall:

Besitzt der Nachbar hingegen genau n Schlüssel, so würde er durch das Abgeben gegen die B-Baum Regeln verstoßen. Man muss nun zwei Knoten zu einem verschmelzen (Konkatenation), wobei ein Element aus dem eine Ebene höher gelegenen Knoten ebenfalls mit einbezogen wird und somit ein Eintrag der Größe $2n$ entsteht (dabei sind die Nachbarkriterien aus Fall 1 wieder gültig). Dabei kann es nun dazu kommen, dass der Vaterknoten durch das Abgeben zu wenige Schlüssel enthält und somit die Ausgleichsoperationen wiederholt werden müssen. Das kann sich bis zur Wurzel fortsetzen, wobei der Baum dann um eine Ebene schrumpfen würde.

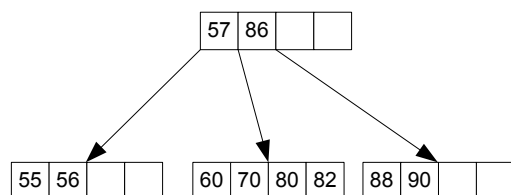
Ein weiteres Beispiel soll das illustrieren:



(Abb. 9: Löschen des Schlüssels 84)

Abb. 9 zeigt das Löschen des Schlüssels 84, ein Unterlauf entsteht; Ausgleichsoperationen wie in Fall 1 beschrieben, greifen hier nicht, da alle Blätter nur n Einträge haben. Wie man sieht, werden die beiden rechten Knoten zu einem verschmolzen, wobei der größte Schlüssel (allgemein der Eintrag, von dem aus Verweise auf die betroffenen Nachfolger zeigen) des Vaterknotens in den Prozess miteinbezogen wird; er ist quasi der Trenneintrag zwischen den Schlüsseln des alten linken und des alten rechten Knotens.

Alternativ zum obigen Vorgehen hätte man auch folgenden Verknüpfungsvorgang wählen können:



(Abb. 10: Alternative Vorgehensweise beim Löschen der 84)

In Abb. 10 wurden die beiden Knoten in der Mitte der untersten Ebene (siehe Abb. 9 – Bild rechts oben) zu einem verschmolzen. Die 80 (der Schlüssel, der zwischen den Verweisen stand, die auf die zu verschmelzenden Knoten hinweisen) wurde aus dem Vaterknoten in das Blatt verschoben. Die B-Baum Eigenschaften werden erfüllt.

3.3.2. Löschen in einem inneren Knoten

Bei inneren Knoten sind die Verhältnisse komplizierter. So kann es auch bei diesem Löschvorgang wieder vorkommen, dass man einen Unterlauf erhält, dessen Behandlung man nicht vergessen darf.

Man sucht zunächst den zu entfernenden Eintrag nach der in 3.1. beschriebenen Methode.

Anschließend folgt man dem rechts liegenden Verweis im betreffenden Knoten; auf der Blattebene (wo man das Löschen wie in 3.3.1. beschrieben durchführen kann) verschiebt man dann den am weitesten links gelegenen Schlüssel (also den kleinsten) an die Stelle des zu löschenden Elements.

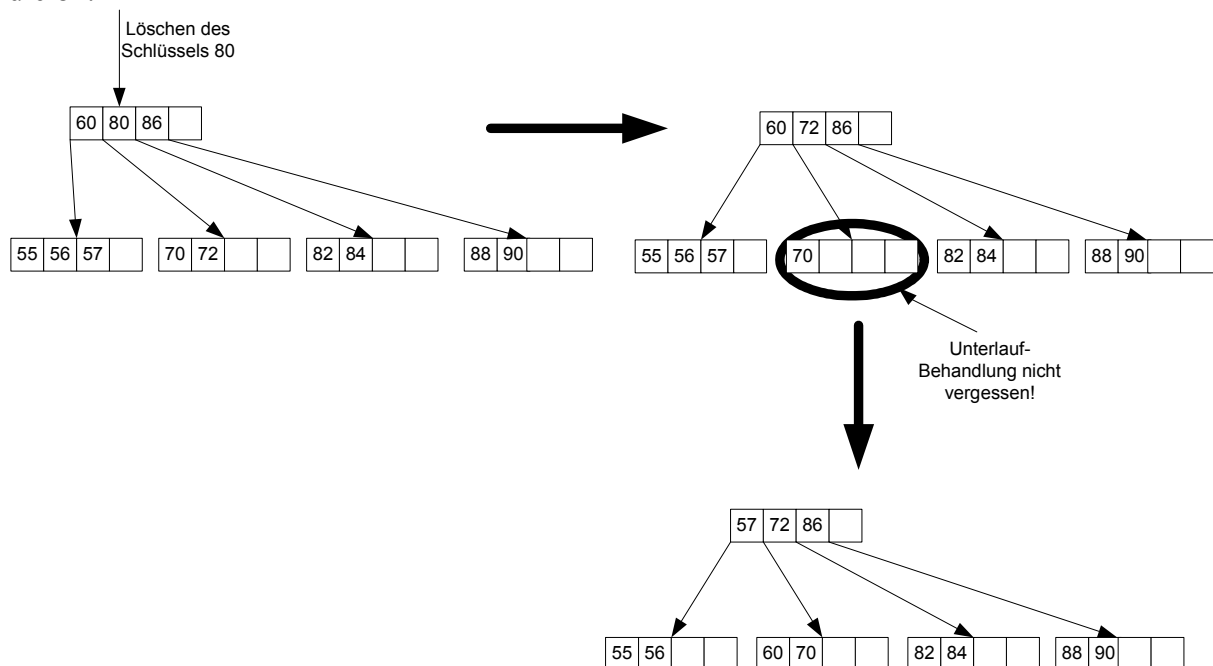
Alternativ dazu kann man auch dem links gelegenen Verweis folgen. Auf Blattebene angekommen, muss man dann allerdings das am weitesten rechts gelegene Element nach oben verschieben.

Am Beispiel erkennt man am besten, wie eine Löschoperation im inneren Knoten konkret aussieht (hier soll der obige alternative zweite Fall gezeigt werden):

An die Stelle der 80 rückt man den Schlüssel aus dem Knoten auf den der linke Pointer des Schlüssels 80 zeigte und der am größten ist, hier also die 72.

Den Unterlauf im zweiten Knoten von links in Abb. 11 darf man nicht übersehen. Man verwendet dann die oben beschriebenen Ausgleichsverfahren.

Es wäre ebenso möglich gewesen, dem rechten Pointer zu folgen und dann den kleinsten Schlüssel des entsprechenden Knotens eine Stufe nach oben zu befördern, im konkreten Fall die 82.



(Abb. 11: Löschen der 80 und anschließende Unterlaufbehandlung im Blatt)

Abschließend sei nochmals angemerkt, dass Löschoperationen der einzige Weg sind, einen B-Baum um eine Ebene schrumpfen zu lassen; das ist genau dann der Fall wenn sich Verschmelzungsvorgänge bis hin zur Wurzel fortsetzen

4. Varianten des B-Baums

4.1. B*-Bäume

Die wesentliche Veränderung gegenüber dem normalen B-Baum ist, dass die Knoten nunmehr zu $2/3$ gefüllt sein müssen; beim B-Baum ist bereits eine Belegung von 50% ausreichend.

Um zu einer $2/3$ Belegung der Knoten zu gelangen, bedarf es einiger Änderungen an der Splitting-Regelung beim Einfügen; es soll vermieden werden, dass Knoten allzu oft aufgespaltet werden. Exakte Details dazu finden sich bei [Knuth], S.487ff.

Der Grundgedanke ist, dass man ein lokales Umverteilungsschema verwendet, um somit die Spaltung so lange zu vermeiden, bis zwei Nachfolgerknoten voll sind. Man teilt dann diese zwei Knoten in drei auf, wobei dann jeder von ihnen zu 66% voll ist. Die Veränderungen am Algorithmus zur Einhaltung der B-Baum Kriterien halten sich in Grenzen; der Vorteil ist aber, dass die Höhe des Baums geringer wird (häufiges Spalten der Knoten das bis zur Wurzel vordringt vergrößert ja die Höhe des Baums) und somit eine schnellere Suche möglich ist (vgl. [Comer], S.129).

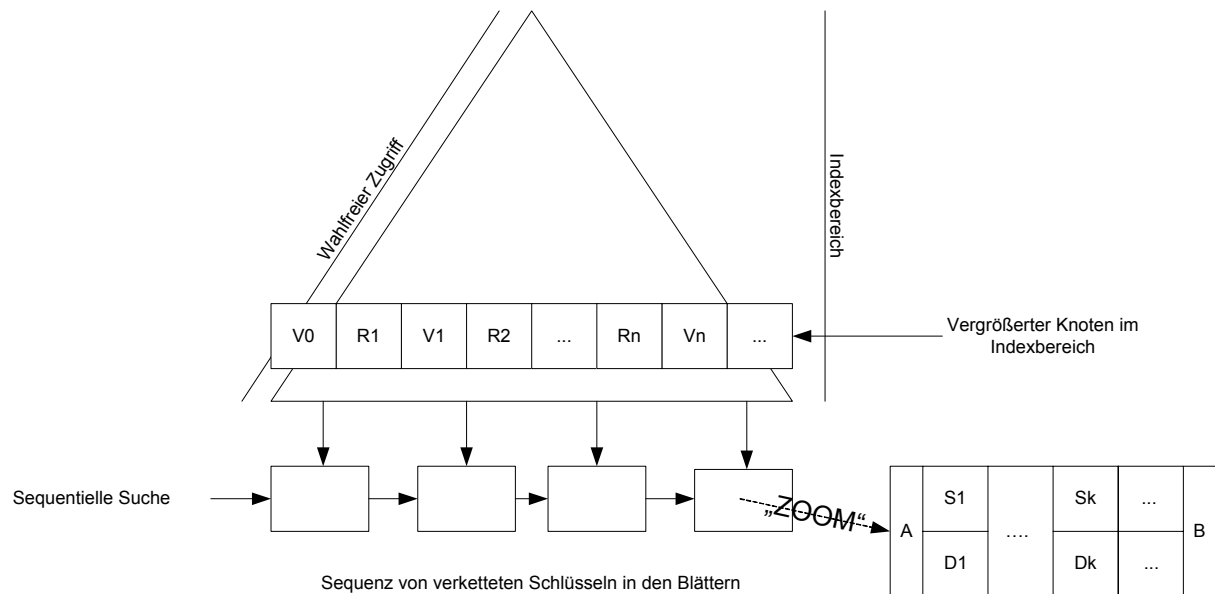
4.2. B+-Bäume

Das entscheidende Merkmal eines B+-Baums (den man auch „hohlen Baum“ nennt) ist, dass alle Datensätze (siehe Vergrößerung eines B-Baum Knotens in Abb. 2) in den Blättern liegen, die ursprünglichen Charakteristika des normalen B-Baums bleiben jedoch erhalten; die Wege von der Wurzel zu den Blättern stets gleich lang. Da die Blattknoten miteinander verknüpft sind und einer verketteten Liste ähneln, eignen sich B+-Bäume gut zur sequentiellen Abarbeitung.

In den inneren Knoten eines B+-Baums liegen nur noch Referenzschlüssel, die [Comer], S.129, auch „roadmap“ nennt. Um einen gesuchten Wert zu finden, muss man nun durch die Indizes bis zur Blattebene vordringen, wobei die Werte im Indexbereich keine Rolle spielen, da man den vollen Datensatz nur auf der Blattebene findet.

Abb. 12 zeigt vergrößerte Knoten sowohl im Index- als auch im Blattbereich. Man erkennt die Verkettung im Blattbereich mittels Zeiger A und B die auf den Vorgänger respektive Nachfolger zeigen (schnelle sequentielle Abarbeitung).

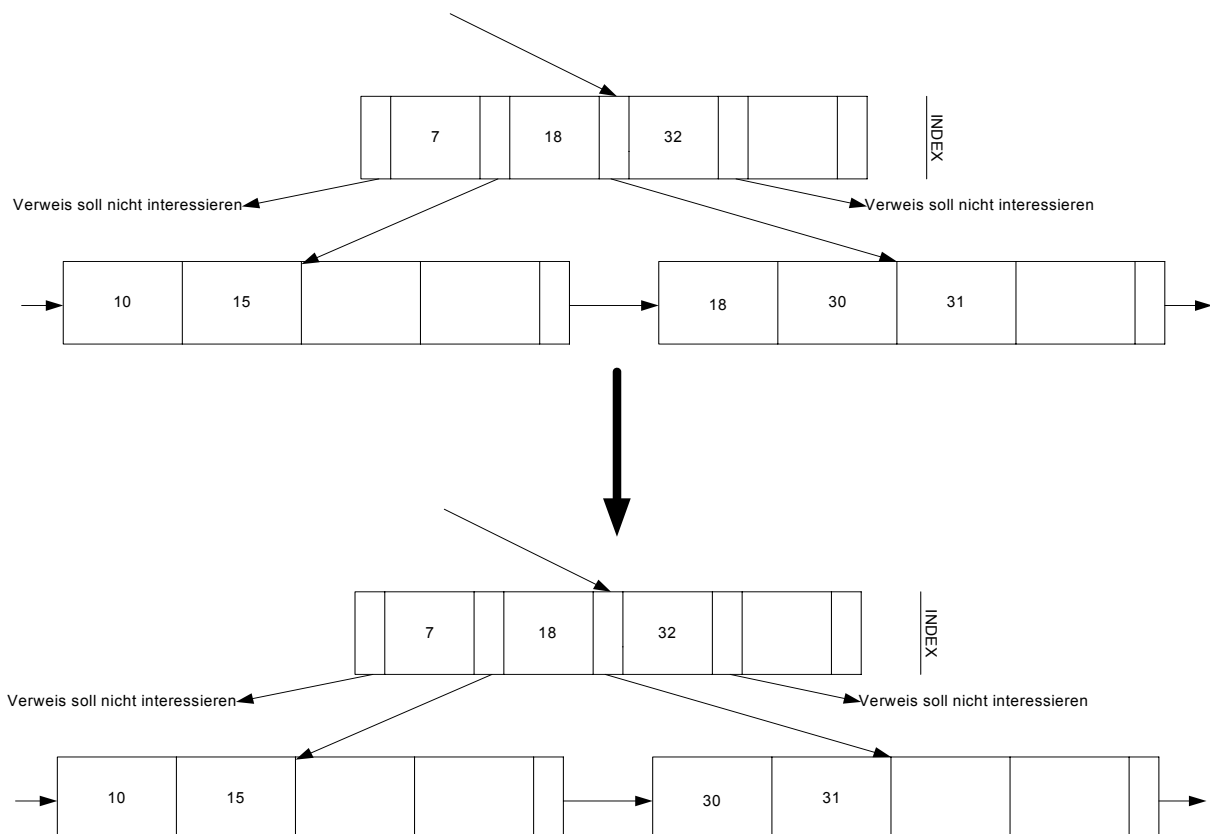
Wie man sieht, sind im Indexbereich nur noch Verweise V und Referenzschlüssel R gespeichert.



(Abb. 12: B+-Baum, vgl. auch [Kemper], S. 212)

Der Löschvorgang vereinfacht sich durch das ausschließliche Vorliegen von Datensätzen in Blättern, da man jetzt auf aufwändige Umstrukturierungsoperationen verzichten kann, solange die Blattknoten mindestens bis zur Hälfte gefüllt sind. Tritt dennoch ein Unterlauf auf, wird man auf die Umverteilungs- oder Verkettungsoperationen (wie sie in Kapitel 3 erläutert wurden) zurückgreifen müssen um so die Kriterien im Indexbereich und den Blättern einhalten zu können.

Ein Beispiel zur Löschoperation soll die Besonderheit der Trennung in Index und eigentlichen Datenbereich veranschaulichen:



(Abb. 13: Löschen im B+-Baum)

Löscht man nun etwa die 18, so verschwindet sie zwar aus dem Blatt, sie bleibt jedoch im Index enthalten, wo sie weiter als Trennwert fungiert:

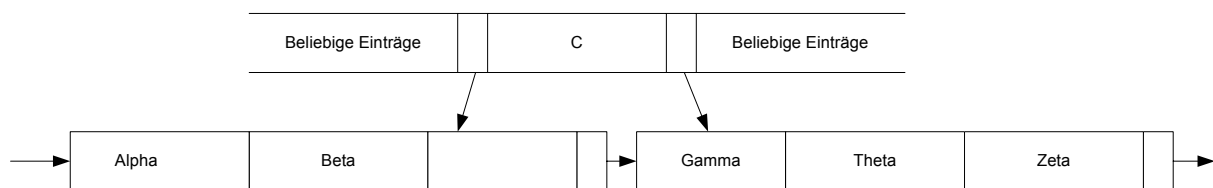
Das Einfügen verläuft ähnlich zum Verfahren beim Standard B-Baum. Spaltet man jedoch ein Blatt auf, so wird man nicht den mittleren Schlüssel eine Ebene nach oben schieben, sondern nur eine Kopie davon; der betreffende Schlüssel bleibt im Blatt (vgl. [Comer] S.130).

4.3. Präfix B+-Bäume

Um Speicherplatz zu sparen, führte man Präfix B+-Bäume ein. Die Besonderheit hierbei ist, dass man anstelle von vollständigen (String-)Schlüsseln einfach kürzere Zeichenfolgen oder gar nur ein Zeichen zur Unterscheidung verwendet, was folgendes Beispiel in Abb. 14 zeigt. Man verwendet hier den kürzesten Wert um Platz zu sparen, also einen Buchstaben.

Somit kann man mehr Schlüssel in einen Knoten unterbringen wodurch die Zahl der Verzweigungen ansteigt und die Höhe des Baums abnimmt. Wie schon erwähnt, spart man dadurch nicht nur Speicherplatz, sondern man erhält auch niedrigere Zugriffszeiten.

Ein Beispiel soll diese Baumart veranschaulichen:



(Abb. 14: Präfix B+-Baum)

Es genügt bereits das Zeichen „c“ des Index um die Sequenzen „Alpha, Beta“ und „Gamma, Theta, Zeta“ von einander zu trennen.

5. Mehrbenutzersysteme und B-Bäume

Ein großer Einsatzbereich für B-Bäume sind Datenbanksysteme, an denen normalerweise viele Benutzer gleichzeitig arbeiten. Wenn mehrere Benutzer simultan an einem System arbeiten, führt das unter Umständen zu Schwierigkeiten; ein User liest z.B. eine Seite von der Platte ein und will die in der Seite gespeicherten Datensätze für weitere Berechnungen verwenden. Zur gleichen Zeit verändert aber ein anderer Benutzer, der vom Lesevorgang des anderen nichts weiß, eben diese Seite, so dass die gespeicherten Daten nicht mehr aktuell sind und der User, der die Berechnung durchführen will, arbeitet nun mit veralteten Informationen. Es bedarf nun einiger Vorschriften und sog. „Sperrern“ (engl. „locks“) um vernünftig (ohne Inkonsistenzen und innerhalb angemessener Zeitspannen) in einer komplexen Umgebung mit mehreren Benutzern arbeiten zu können. Situationen, in denen sich mehrere Benutzer gegenseitig in aussichtslose „Verklemmungen“ (engl. „deadlock“) bringen können, sollen ausgeschlossen werden können.

So wird etwa eine Suchoperation einen Knoten sperren, damit keine konkurrierende Instanz darauf zugreifen kann. Geht der Suchprozess dann eine Ebene weiter, so wird die Sperre wieder aufgehoben und andere Benutzer können darauf zugreifen. Gleichzeitig sind somit maximal zwei Knoten gesperrt (in dem Moment in dem die Suche eine Ebene voranschreitet ist der vorhergehende Knoten ja noch kurz gesperrt).

Komplizierter ist das Vorgehen bei Einfüge- oder Löschvorgängen (später kurz Aktualisierungsprozess); bei diesen Operationen können infolge der Umstrukturierungsvorgänge im Baum (um die Eigenschaften E1-E6 zu erfüllen) mehrere Ebenen betroffen sein.

Um Probleme zu vermeiden, reserviert der Aktualisierungsprozess alle Knoten die er aufsucht. Stellt der Prozess dann später fest, dass vorher reservierte Knoten tatsächlich geändert werden müssen (etwa beim Verschmelzen von Knoten, das sich über mehrere Ebenen erstreckt), so wird die Reservierung in eine Sperre umgewandelt.

Reservierte Knoten kann man aber dennoch lesen.

Ist die Aktualisierung abgeschlossen, so werden die Sperren und Reservierungen aufgehoben.

6. Abschließende Bewertung des B-Baums

B-Bäume weisen eine geringe Höhe auf, folglich lassen sie sich rasch durchsuchen, was besonders in Hinblick auf die Anzahl von Massenspeicherzugriffen (die ja vergleichsweise langsam sind) wichtig ist, sie werden nämlich reduziert.

Die Methoden zum Einfügen und Löschen von Daten sind verhältnismäßig gut und einfach zu implementieren.

Es kann vorkommen (häufiges Spalten eines Knotens), dass ein Knoten „nur“ zu 50% belegt ist und somit Speicherplatz vergeudet wird; andererseits mag mancher dies als Vorteil empfinden, schließlich ist eine Belegung von 50% garantiert. Wie gezeigt, lässt sich dieses Problem durch Verfeinerungen in Form von B*-Bäumen mindern.

7. Literaturverzeichnis

- [Bayer] Bayer, R., McCreight, E. „Organization and Maintenance of Large Ordered Indexes“, *Acta Informatica* 1, 3 (1972), 173-189
- [Comer] Comer, D. „The Ubiquitous B-Tree“, *ACM Computing Surveys* 11, 2 (1979), 121-137
- [Cormen] Cormen, Th.H., Leiserson Ch.E., Rivest, R.L., Stein C., *Introduction to algorithms. Second edition*, MIT Press, Cambridge MA, 2001
- [Kemper] Kemper, A., Eickler, A., *Datenbanksysteme. Eine Einführung*, Oldenbourg, München, 2001
- [Knuth] Knuth D.E., *The Art of Computer Programming. Volume 3. Sorting and Searching*, Addison Wesley Longman, Reading MA, 1998

Zusätzliche Informationen können auch dem Begleitmaterial zur Vorlesung „Praktische Informatik I“ (gehalten von Professor A. Kemper, Ph.D. im Wintersemester 2002/2003) entnommen werden.

Zu finden sind die Unterlagen unter folgender URL:

<http://www.db.fmi.uni-passau.de/uni/WS02-03/PI1/vorlesung/Unterlagen/>

Unter der URL

http://www.fmi.uni-passau.de/~bachmail/Proseminar/Bayer_Originalarbeit.pdf

findet man die Originalarbeit „*Organization and Maintenance of Large Ordered Indexes*“ von Bayer und McCreight ([Bayer]), in der B-Bäume eingeführt wurden.

Die empfehlenswerte Arbeit von [Comer] findet man unter

http://www.fmi.uni-passau.de/~bachmail/Proseminar/ubiquitous_btree.pdf