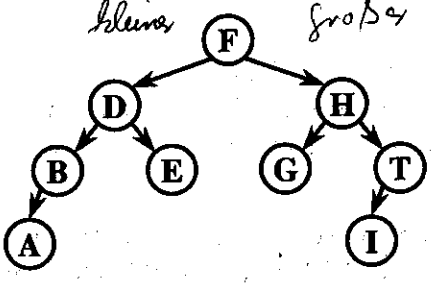
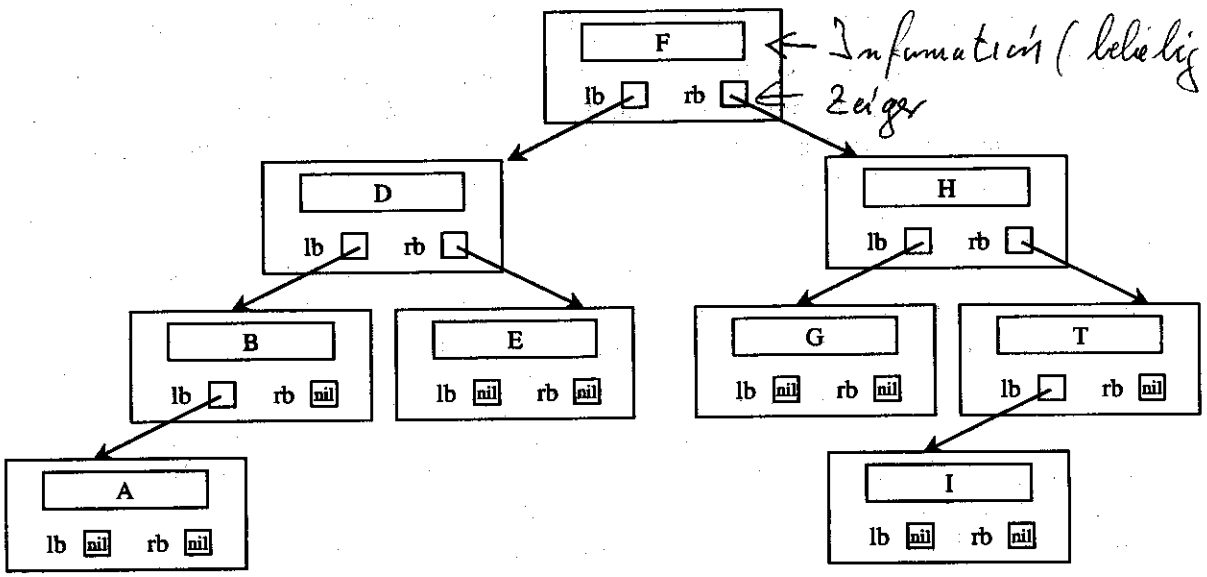
2.2. Darstellung im Speicher

Der Binärbaum:



kann im Speicher wie folgt dargestellt werden:



In den meisten Algorithmen für balancierte Binärbäume hat jeder Knoten vier Felder: die Pointer auf den linken und den rechten Sohn, ein Balancefeld, das vom Algorithmus abhängt, und ein Datenfeld.

Auf einen **externen** Knoten verweist ein NIL-Pointer seines Vaters, seine Felder sind bedeutungslos. In binären Suchbäumen ist ein Datenfeld irgendeine sortierbare Information, so daß für alle **internen** Knoten „p" gilt: alle Werte im linken Teilbaum des Knotens „p" sind kleiner und alle Knoten im rechten größer als der Schlüssel von „p".

2.3. Suchen, Einfügen, Löschen

Die Konstruktion eines zufälligen Binärsuchbaums, also ohne Balancierung, ist trivial. Der Schlüssel des einzufügenden Knotens, der notwendigerweise ein Blatt ist, wird mit dem Schlüssel der Wurzel verglichen. Ist er kleiner, fährt die Funktion rekursiv mit dem linken Unterbaum fort, andernfalls mit dem rechten, bis sie auf einen externen Knoten trifft. Die so gefundene Folge von Knoten, die von der Wurzel bis zu einem externen Knoten führt, heißt Suchpfad.

Der gefundene externe Knoten ist der Einfügeplatz des neuen Blatts. Bei seiner Initialisierung werden die Pointer auf die Söhne auf NIL und das Datenfeld auf den Schlüssel gesetzt. Der unmittelbar vorhergehende interne Knoten ist der Vater, bei dem der Verweis auf den neuen Sohn aktualisiert werden muß.

Das Löschen eines Knotens erweist sich als etwas komplizierter: wie im Fall des Einfügens vergleicht man den zu löschenden Schlüssel mit der Wurzel und folgt dann dem Suchpfad. Ist der zu löschende Knoten gefunden, gilt es festzustellen, ob er zwei Söhne hat. Trifft das nicht zu, so wird der auf ihn zeigende Pointer des Vaters entweder auf NIL oder auf den

einzigen Sohn gesetzt. Hat er allerdings zwei Söhne, so ist sein Wert (Inhalt des Datenfelds) durch den größten des linken Teilbaums oder den kleinsten des rechten Teilbaums zu ersetzen.

Binärbäume lohnen sich im wesentlichen nur für eine große Anzahl von Elementen, jedoch sollte diese auch nicht zu groß sein, da sich die Datenstruktur in erster Linie für die interne Speicherung der Daten eignet. Auf externen Massenspeichern verwendet man besser beispielsweise B-Bäume.

Ungewißheit besteht bei der Verwendung von zufälligen Binärbäumen darüber, wie die Algorithmen zum Einfügen und Löschen die Struktur des Binärbaums im Laufe der Zeit verändern. Der schlechteste Fall ist offensichtlich die lineare Liste, in der die Elemente zum Beispiel in auf- oder absteigend sortierter Reihenfolge eingefügt werden.

Allgemeine Suchbäume gestatten es, die Operationen Suchen, Einfügen und Löschen im Mittel in der Zeit O(ld N) durchzuführen, wobei n die Anzahl der Knoten des Suchbaumes ist. Voraussetzung für dieses günstige Zeitverhalten ist, daß die Knoten einigermaßen gleichmäßig auf alle Unterbäume des Baums verteilt sind.

Bei dynamischer Veränderung durch Einfüge- und Löschoperationen kann der Baum jedoch zunehmend entarten, also z.B. "linkslastig" oder "rechtslastig" werden. Bei ausgeglichenen Bäumen wird diese Degeneration durch ständige Korrektur des Baumes vermieden. Daher garantieren ausgeglichene Bäume eine Such- und Einfügezeit von O(ld N)

Der zusätzliche Aufwand zur Neuorganisation des Baums lohnt aber meist nur, wenn die Anzahl der Zugriffe bedeutend größer als die Anzahl der Einfüge- und Löschoperationen ist. Trotz ihres Risikos, zur linearen Liste zu degenerieren, liefern zufällige Binärbäume meist befriedigende Leistungen, da sie sich durch sehr einfache und kurze Implementierungen auszeichnen.

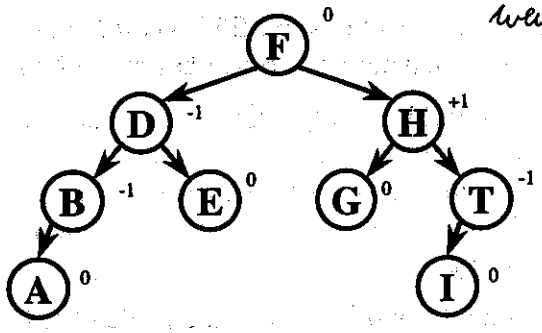
2.4. AVL-Bäume

Prozeduren zum Einfügen und Löschen, die jedesmal die perfekte Balance des Baums wiederherstellen, lohnen sich also kaum. Daher muß man versuchen, durch schwächere Definitionen der Balance Verbesserungen zu erzielen. Ein solches Kriterium für unvollständige Balance sollte einfache Prozeduren zur Neuorganisation, auf Kosten einer nur geringen Verschlechterung der mittleren Pfadlänge gegenüber dem Optimum, ermöglichen.

Eine adäquate Definition dafür wurde von den beiden russischen Mathematikern Adelson-Velskii und Landis 1961 postuliert: ein Baum ist genau dann ausgeglichen, wenn sich für jeden Knoten die Höhen der zugehörigen Teilbäume um höchstens 1 unterscheiden.

Bäume, die diese Bedingung erfüllen, heißen nach ihren Schöpfern AVL- oder auch höhenbalancierte Bäume. Die Definition ist einfach, führt aber dennoch zu sehr handlichen Ausgleichsprozeduren und zu Pfadlängen die praktisch mit denen in einem perfekt balancierten Baum identisch sind.

Das Balancefeld eines AVL-Knotens kann die Höhendifferenz mit zwei Bits anzeigen (+1: der rechte Sohn ist höher, 0: gleiche Höhe, -1: der linke Sohn ist höher). Knoten mit einem Balancegrad von 0 bezeichnet man als balancierte Knoten, andere als unbalancierte.



weil F-D-B-A
= F-H-T-I
= 0

Man kann beweisen, daß ein AVL-Baum mit N Knoten höchstens die Höhe 1,4404*ld (N+2) besitzt, also höchstens 44% über dem minimalen Wert ld N. AVL-Bäume eignen sich daher zur Verwaltung von Daten, in denen oft gesucht wird und die oft verändert werden.

2.5. B-Bäume

kein Binär-Baum!
= balanced

Für die Speicherung auf externen Speichern verwendet man besser B-Bäume, um die Anzahl der Zugriffe zu verringern. Man kann dazu die Suchbaumstruktur sinngemäß auf die Verwendung externer Speichereinheiten übertragen. Z.B. sind für Magnetplattenspeicher die Zeiger auf die Söhne eines Knotens dann nicht mehr Hauptspeicheradressen, sondern Plattenspeicheradressen.

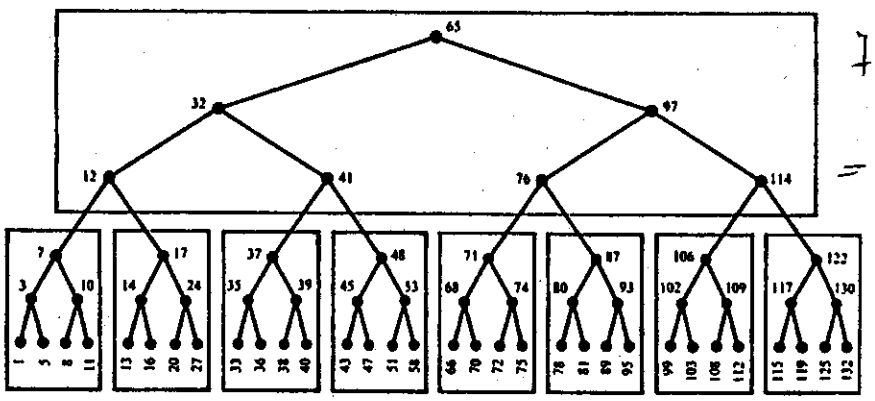
Beispiel: Die Suchzeit, die sich zum Auffinden gewisser Daten ergibt, soll an einem Suchbaum mit 1.000.000 Knoten erläutert werden. Die zu erwartende Suchzeit in einem Suchbaum mit n Knoten beträgt mindestens ld (n) Schritte. Man muß daher mit [ld (1.000.000)] = 20 Plattenzugriffen rechnen.

Man reduziert daher die Plattenzugriffe auf ein Mindestmaß, indem man je Zugriff nicht einen einzelnen Knoten, sondern ein ganzes Teilstück von der Magnetplatte in den Hauptspeicher überträgt und innerhalb dieses Teilstücks sucht.

Teilt man den Baum in die umrandeten Stücke zu je 7 Knoten und überträgt mit jedem Plattenzugriff ein solches Stück in den Hauptspeicher, so reduziert sich die Zahl der Plattenzugriffe für die Suche eines Knotens von maximal 6 auf maximal 2. Bei 1.000.000 Knoten benötigt man also nur noch [$\log_8$ (1.000.000)] = 7 Zugriffe.

7+1

Grad 4-7



7 Schlüssel pro Knoten !!
= 8 Nachfolger

Abb. 1: Aufteilung eines Baumes in Teilstücke zu je 7 Knoten

In der Praxis unterteilt man den Suchbaum (statt in 7 Teilstücke) meistens in Teilstücke der Größe 255 bis 1023 (2^8-1 bis 2^{10}-1) Knoten. Bei einer Stückgröße von 255 Knoten benötigt man für die Suche eines Knotens in einem Baum mit 1.000.000 Knoten

[$\log_{256}$ (1.000.000)] = 2,5 Plattenzugriffe.

Die Suchzeit innerhalb eines Teilstücks mit 255 Knoten, der sich im Hauptspeicher befindet, kann gegenüber dem Plattenzugriff vernachlässigt werden.

Da umfangreiche Einfüge- und Löschoperationen von Knoten zur Entartung eines Baumes, also zu ausgedünnten, listenartigen Strukturen führen können, wurde 1972 von R. Bayer und E. McCreight analog zum ausgeglichenen Baum der B-Baum entwickelt. „B" weist auf "balanced" hin (B-Bäume sind höhen-balancierte Bäume, bei denen alle Blätter auf dem gleichen Niveau liegen).

Ein Baum heißt B-Baum der Ordnung m, falls die folgenden Eigenschaften zutreffen:

a) Jeder Knoten enthält höchstens 2m Schlüssel.

b) Jeder Knoten (mit Ausnahme der Wurzel) enthält mindestens m Schlüssel.

c) Ein Knoten mit k Schlüsseln hat genau k + 1 Söhne oder keinen Sohn.

d) Alle Knoten, die keine Söhne haben, befinden sich auf dem gleichen Niveau.

e) Suchbaumeigenschaft: Sind s_1, ..., s_k mit m <= k <= 2m die Schlüssel eines Knotens x, dann sind alle Schlüssel des ersten Sohnes von x kleiner als s, alle Schlüssel des (k+1)-ten Sohnes größer als s_k und alle Schlüssel des i-ten Sohnes, 1 < i < k+1, größer als s_{i-1}, und kleiner als s_i

2.5.1. B-Baum als Index einer Datenbank

Um eine Anfrage so schnell wie möglich zu beantworten, sollte eine Datenbank möglichst nur die tatsächlich benötigten Datensätze von der Festplatte laden. Dazu bedienen sich Datenbanken so genannter Indexierungstechniken. Zusätzlich zu den eigentlichen Nutzdaten speichern sie einen oder mehrere Indexe, über die sie sehr effizient nach bestimmten Schlüsseln suchen oder sortieren können. Der B-Baum und seine Varianten sind ein De-facto-Standard zur Indexierung von Datenbanken.

Beispielsweise lassen sich mit einem Verzweigungsgrad von 256 Nachfolgern pro Knoten und einer Höhe von 3 bereits 256^3 = 16.777.216 Blätter verwalten. Um einen Datensatz zu suchen, genügen drei Blockzugriffe (ca. 30 ms). Ein sequenzieller Scan über eine Datenbank dieser Größe würde demgegenüber knapp 20 Minuten benötigen (bei einer Blockgröße von 2 KByte und einer Übertragungsrate von 30 MByte/s).

Die eigentlichen Datensätze bringen aktuelle Datenbanken übrigens nur in den Blättern unter. Die darüberliegenden inneren Knoten speichern lediglich Schlüssel und Zeiger auf Nachfolgeknoten. Dies spart Platz in den Indexknoten und trägt so zu einem hohen Verzweigungsgrad bei. Diese Variante des B-Baums heißt in der Literatur auch B*-Baum.

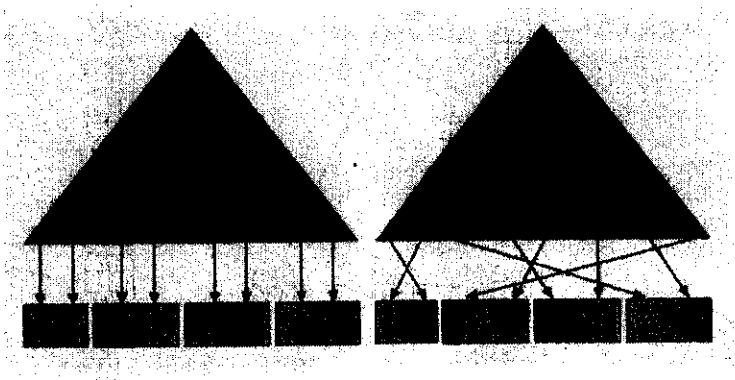
In den Blättern liegen die Datensätze in der Reihenfolge der Schlüssel vor. Hat man erst einmal einen Datensatz über den Index gesucht und den entsprechenden Datenblock in den Speicher geladen, so sind damit automatisch auch die restlichen, unmittelbar folgenden Datensätze des Blocks im Zugriff. Durch diese auch als Satz- Clustering bezeichnete Eigenschaft lassen sich die Datensätze effizient in Schlüsselreihenfolge sequenziell durchlaufen. Das soeben Gesagte gilt allerdings nur für den so genannten Primärindex.

Meist benötigt man in einer Datenbank jedoch auch die Möglichkeit, nach weiteren Schlüsseln zu suchen. Dies lässt sich leicht durch zusätzliche, so genannte Sekundärindexe erreichen. Dabei handelt es sich ebenfalls um B(*)- Bäume, in deren Blättern jedoch nicht die vollständigen Datensätze, sondern lediglich Verweise auf diese stehen.

Beim Suchen über einen Sekundärindex fällt also ein zusätzlicher Blockzugriff an, um den eigentlichen Datensatz zu laden. Bei einer einzelnen Punktsuche fällt dies nicht weiter ins Gewicht, wohl aber beim sequenziellen Durchlaufen der Datenbank. Bezüglich eines Se-

kundärschlüssels liegen die Datensätze nämlich ungeclustert, also in zufälliger Reihenfolge auf der Platte, sodass für jeden einzelnen Datensatz ein erneuter Plattenzugriff anfällt.

Daher sind Sekundärindexe bei Anfragen mit großen Ergebnismengen spürbar langsamer als der Primärindex. Da die Daten nur in einer Reihenfolge auf der Platte liegen können, kann es nur einen Primärindex geben. Die Anzahl der Sekundärindexe ist dagegen beliebig.



Primitive PC-„Datenbanken" wie etwa die alten dBASE- Versionen legen die Daten übrigens grundsätzlich ungeclustert ab, haben also überhaupt keinen Primärindex im Sinne der obigen Erklärung, sondern alle Indexe sind Sekundärindexe.

Nicht zu verwechseln sind diese beiden Indexbegriffe mit den im Datenbankbereich ebenfalls gebräuchlichen Begriffen des Primär- und Sekundärschlüssels. Ein Primärschlüssel einer Tabelle hat die Eigenschaft, dass jeder Schlüsselwert einen Datensatz eindeutig charakterisiert. Gleiche Werte dürfen innerhalb einer Tabelle also nicht mehrfach vorkommen.

Sekundärschlüssel haben diese Einschränkung nicht. Ein typischer Primärschlüssel ist zum Beispiel eine fortlaufende Kundennummer (keine zwei Kunden haben dieselbe Nummer). In Feldern wie Name oder Vorname kann dagegen durchaus derselbe Wert in verschiedenen Datensätzen auftreten, weswegen sie nur als Sekundärschlüssel taugen.

Trotzdem kann man beispielsweise einen Sekundärschlüssel wie den Namen für den Primärindex benutzen; dies bedeutet lediglich, dass die Daten auf der Platte entsprechend diesem Schlüssel geclustert sind.

2.5.2. Beispiele

1) Bei der graphischen Darstellung von B-Bäumen faßt man alle Knoten eines Teilstücks in einem Knoten zusammen. Abb. 2a zeigt den Baum aus Abb. 1 als B-Baum der Ordnung 4 (bzw. 5, 6 oder 7).
2) Einen B-Baum der Ordnung 2 zeigt Abb. 2b: Jeder Knoten enthält zwei bis vier Schlüssel und besitzt, sofern er kein Blatt ist, drei bis fünf Nachfolger. Um in einem B-Baum mit n Knoten der Ordnung m einen Wert zu suchen, muß man höchstens $\log_{m+1}(n)$-mal auf die Platte zugreifen.

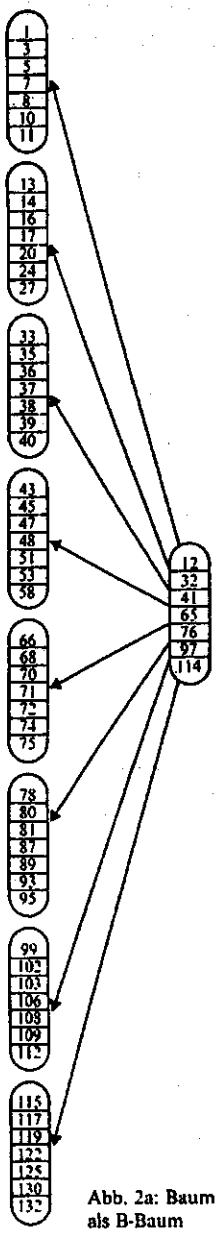


Abb. 2a: Baum aus Abb. 1 als B-Baum

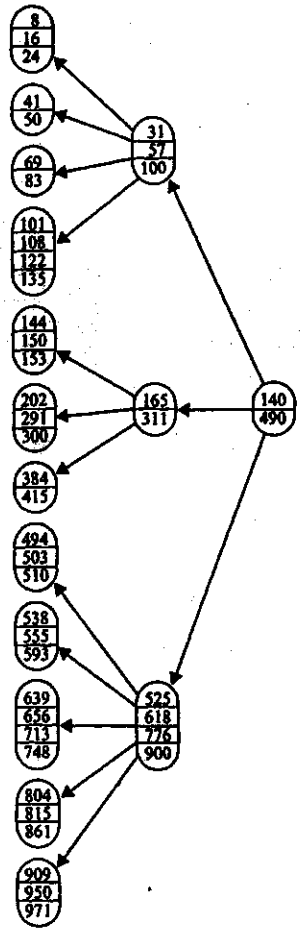


Abb. 2b: B-Baum der Ordnung 2

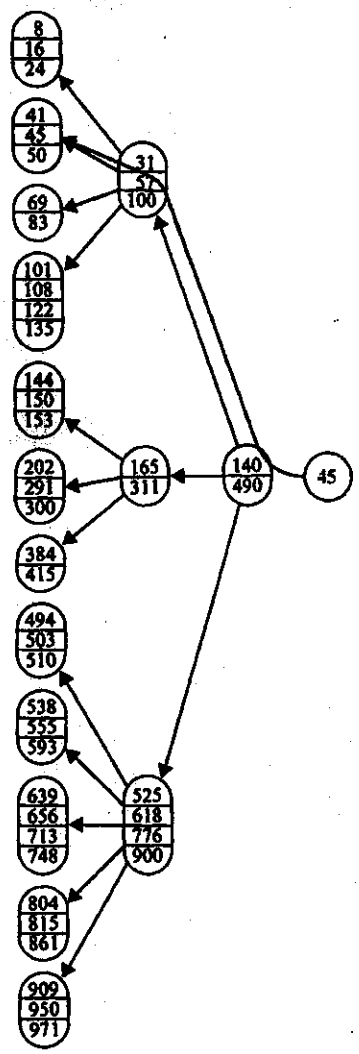


Abb. 3: Einfügen des Schlüssels 45 in den B-Baum von Abb. 2b

B-Bäume sind also sehr gut geeignet für die Verwaltung von Daten, auf die häufig zugegriffen wird und die sich laufend verändern.

Suchen:

Einen Schlüssel s im B-Baum findet man wie bei Suchbäumen, indem man beginnend bei der Wurzel prüft, ob s im gerade betrachteten Knoten x enthalten ist. Falls dies zutrifft, ist man fertig. Andernfalls prüft man, ob x ein Blatt ist. Falls ja, dann endet die Suche erfolglos, falls nein, dann stellt man fest, zwischen welchen Schlüsseln s_{i-1}, und s_i des Knotens x der Schlüssel s liegt (bzw. ob $s < s_1$, oder ob $s > s_k$ ist) und setzt die Suche danach beim Knoten x.sohn[i] fort.

Einfügen:

Das Einfügen von Schlüsseln geschieht grundsätzlich in den Blättern. Man durchläuft mit dem neu einzutragenden Schlüssel den B-Baum wie beim Suchen, bis man auf das Blatt stößt, wo der Schlüssel einzutragen ist (vgl. Abb. 3). Die Zahl der Schlüssel je Knoten (Wurzel ausgenommen) muß zwischen m und 2m liegen. Beträgt sie weniger als 2m, dann ist das Einfügen erfolgreich beendet. Beträgt sie jedoch 2m, so entsteht ein Überlauf, d. h.

das Blatt enthält jetzt 2m+1 Schlüssel. In diesem Fall teilt man den Knoten in zwei Knoten zu je m Schlüsseln. Der mittlere Schlüssel des Knotens wird vom Vater aufgenommen. Falls auch der Vaterknoten "überläuft", muß er ebenfalls in zwei Knoten aufgeteilt und sein mittlerer Schlüssel nach oben an seinen Vaterknoten weitergereicht werden usw.

In Abb. 3 wird in den B-Baum von Abb. 2b der Schlüssel 45 hinzugefügt. Das Einfügen des Schlüssels 680 dagegen bewirkt zweimal einen Überlauf (Abb. 4a bis 4e). Läuft die Wurzel über, dann wird sie aufgespalten und eine neue Wurzel angelegt. In diesem Fall ist der B-Baum um eine Stufe gewachsen. Man beachte, daß B-Bäume im Gegensatz zu ausgeglichenen und anderen Bäumen von den Blättern zur Wurzel hin wachsen.

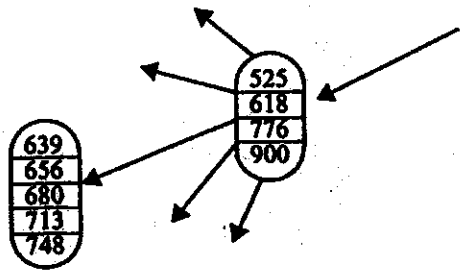


Abb. 4a: Beim Einfügen des Schlüssels 680 in den Baum der Abb. 3 entsteht zunächst diese unzulässige Situation (fünf Schlüssel in einem Knoten, „Überlauf")

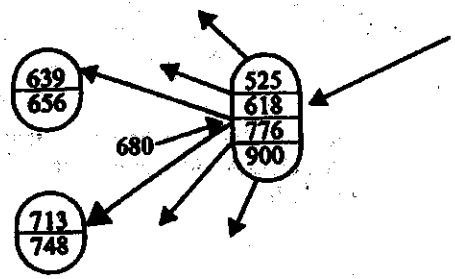


Abb. 4b: Der zu große Knoten wird aufgespalten und der mittlere Schlüssel 680 wird zum Vaterknoten weitergereicht; zugleich ist eine neue Kante zu erzeugen

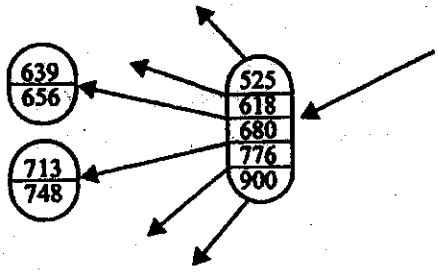


Abb. 4c: Der Vaterknoten enthält nun zu viele Schlüssel. Also muß auch er aufgespalten werden

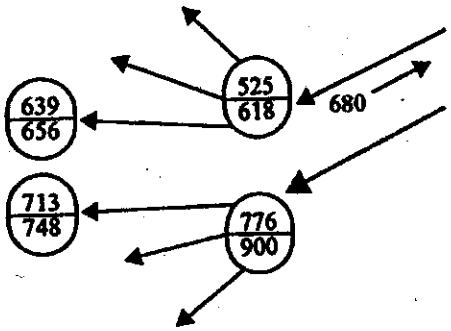


Abb. 4d: Der mittlere Schlüssel (zufällig wiederum 680) wird weitergereicht. Die Endsituation des Gesamtbaums, der sich aus Abb. 3 nach Einfügen von 680 ergibt, ist in Abb. 4e angegeben

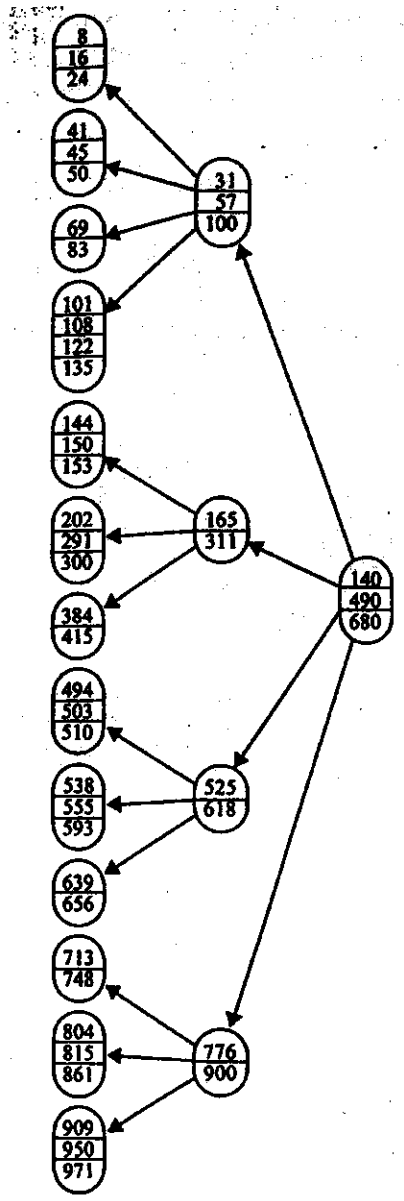


Abb. 4e: B-Baum aus Abb. 3 nach Einfügen von 680

Das Löschen eines Schlüssels s ist komplizierter. Zunächst sucht man den Knoten x, in dem s enthalten ist. Falls x ein Blatt ist, dann löscht man s; eventuell muß man einen „Unterlauf" bereinigen. Falls x kein Blatt ist, dann ermittelt man den nächstgrößeren Schlüssel s' zu s im Baum. s' steht in einem Blatt, und zwar ist s' der äußerste linke Schlüssel im äußersten linken Blatt des Unterbaumes, zu dem die hinter s in x stehende Kante führt.

In Abb. 4e ist z. B. 713 der nächstgrößere Schlüssel zu 680 und 41 der nächstgrößere Schlüssel zu 31 (Abb. 5). Man ersetzt nun s durch s' und löscht s' im Blatt. Ob x ein Blatt ist oder nicht, das Löschen von s führt in jedem Fall zum Löschen eines Schlüssels in einem Blatt x'.

Wenn das Blatt x' weiterhin mindestens m Schlüssel besitzt, ist das Löschen beendet. Stehen in dem Blatt x' dagegen nur noch (m-1) Schlüssel, so spricht man von einem Unterlauf. Man füllt dann x' mit Hilfe der Schlüssel eines Nachbarknotens auf.

Wenn in Abb. 5 z.B. 680 gelöscht werden soll, so wird 680 durch den nächstgrößeren Schlüssel 713 ersetzt. Durch das Löschen von 713 entsteht dann ein Unterlauf (Abb. 6a).

Über den Vaterknoten läßt man nun einen Schlüssel vom Nachbarknoten in das Blatt fließen (hier: 776 fließt in das Blatt und 804 rückt in den Vaterknoten nach, Abb. 6 b).

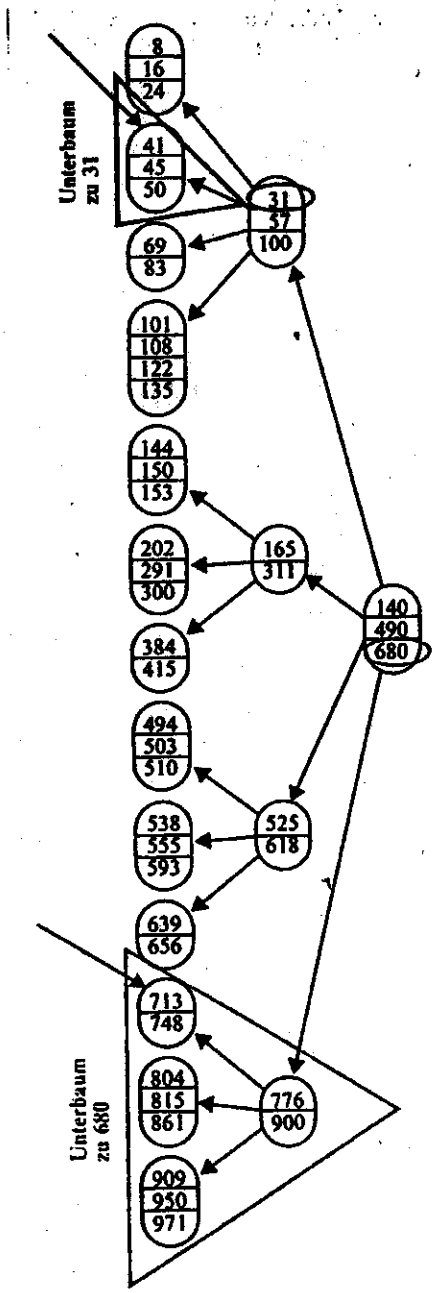


Abb. 5: 41 ist nächstgrößerer Schlüssel zu 31, 713 ist nächstgrößerer Schlüssel zu 680

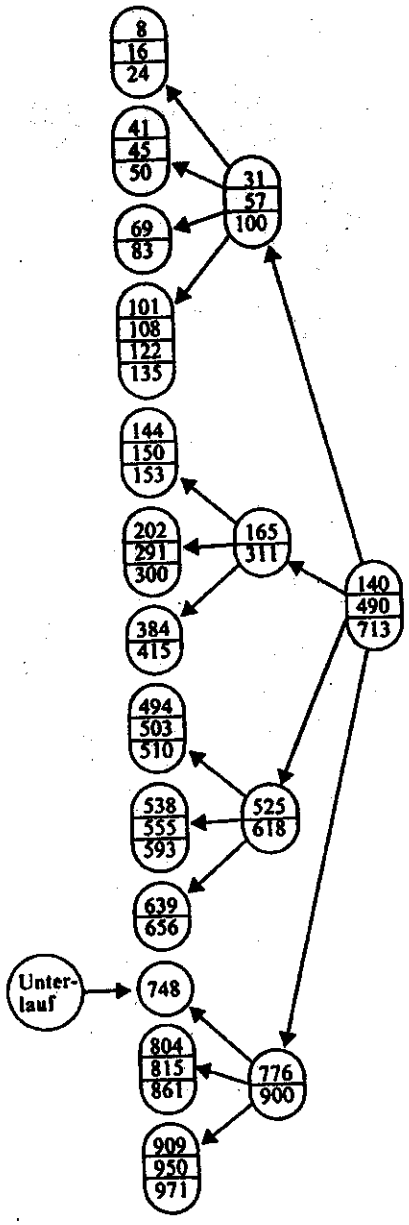


Abb. 6a: Zwischensituation beim Löschen von 680 im Baum der Abb. 5. In einem Blatt ist ein „Unterlauf" entstanden

Falls der Nachbarknoten mindestens (m+1) Schlüssel enthielt, ist nun das Löschen beendet. Besitzt der Nachbarknoten jedoch genau m Schlüssel, dann würde er beim Weiterschieben eines Schlüssels unterlaufen. In diesem Fall verschmilzt man ihn mit dem untergelaufenen Blatt und fügt noch den zugehörigen Schlüssel des Vaterknotens hinzu, so daß ein Blatt mit 2m Schlüsseln entsteht. Löscht man in Abb. 6b z.B. noch 776, dann entsteht die in Abb. 6c angegebene Situation. Da hierbei im Vaterknoten ein Schlüssel entfernt wird, kann dieser eventuell unterlaufen usw.

Löscht man in dem Baum der Abb. 5 nacheinander erst 680 und dann 776, so erhält man schließlich den in Abb. 6d angegebenen B-Baum.

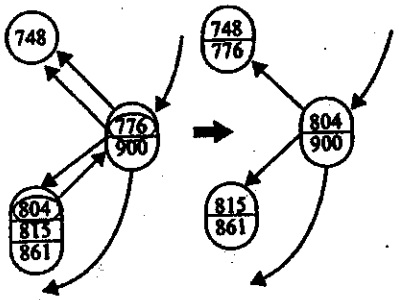


Abb. 6b: Auffüllen eines untergelaufenen Blattes

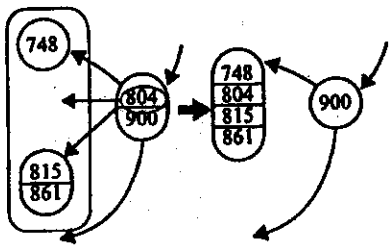


Abb. 6c: Verschmelzen zweier Blätter; in diesem Fall muß der Vaterknoten, der als Folge des Verschmelzens untergelaufen ist, mit seinem Nachbarknoten verschmolzen werden usw.

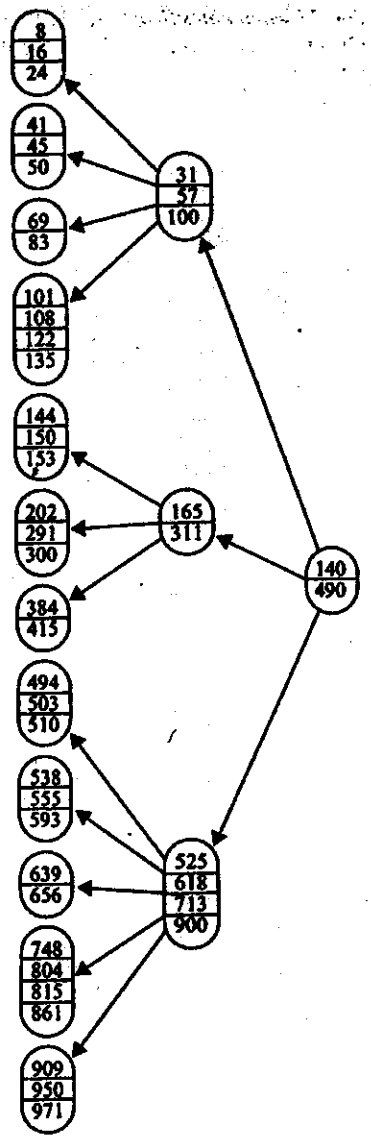


Abb. 6d: B-Baum aus Abb. 5 nach dem Entfernen der Schlüssel 680 und 776.

Man erkennt: Suchen, Einfügen und Löschen erfordert so viele Schritte, wie es Knoten auf dem Pfad von der Wurzel bis zu einem Blatt gibt. Bei einem B-Baum der Ordnung m mit n Knoten sind dies größenordnungsmäßig $\log_m(n)$. B-Bäume sind daher eine ideale Speicherstruktur für sich laufend verändernde Datenbestände.

Bemerkung:

Ein B-Baum der Ordnung m wird oft auch folgendermaßen definiert:

a) Alle Blätter besitzen das gleiche Niveau.
b) Jeder Knoten (mit Ausnahme der Wurzel) besitzt mindestens [m/2] Söhne und höchstens m Söhne.
c) Die Wurzel besitzt mindestens 2 und höchstens m Söhne.
d) Jeder Knoten mit k Söhnen enthält genau k - 1 Schlüssel.
e) Suchbaumeigenschaft wie oben.

Durch diese Definition sind mehr Bäume erfaßt als durch die ursprüngliche. Zum Beispiel lassen sich so die B-Bäume der Ordnung 3, die sogenannten 2-3-Bäume, definieren, was mit der früheren Definition nicht möglich ist, da hier die maximale Anzahl von Schlüsseln in einem Knoten ungerade ist.