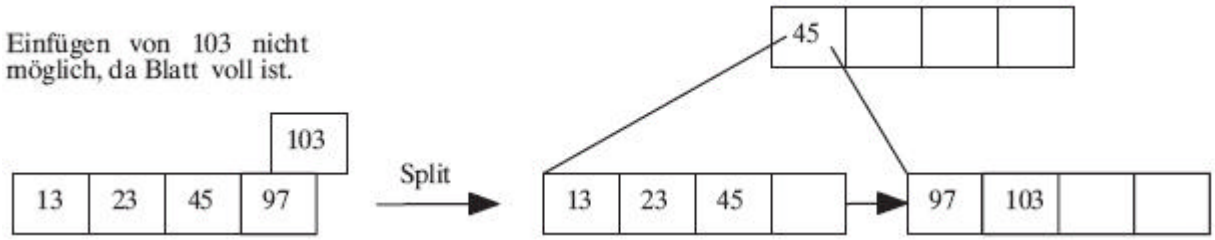
Max-Planck-Institut Saarbrücken
AG5: Datenbanken und Informationssysteme
Dozenten: Prof. Dr.-Ing. G. Weikum, Dr. Ralf Schenkel
Betreuer: Dipl.-Inform. Sergej Sizov, Dipl.-Inform. Christian Zimmer

Informationssysteme (SS 05)
Übungsblatt 9

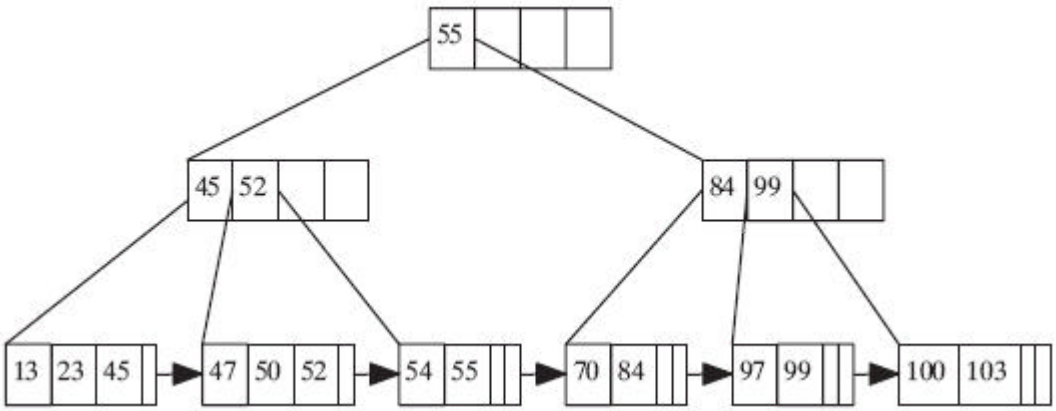
Beispiellösungen

Aufgabe 1: B*-Baum

Die ersten 4 Einfügungen können in ein Blatt erfolgen. Da ein Blatt jedoch höchstens 4 Einträge enthalten kann, muss bei der 5. Einfügeoperation ein ’Split’ durchgeführt werden. Die folgende Abbildung verdeutlicht, wie eine solche Split–Operation vorgeht:



Analog erfolgen die weiteren Einfügungen bis wir zum Schluss den folgenden Baum erhalten (Zu beachten ist, dass beim Split eines Nicht–Blattknotens der Median in den Vaterknoten wandert):



Durch das Löschen von 70 würde ein Blatt den Mindestfüllgrad unterschreiten. Um die Invariante der Definition des B*–Baumes aufrecht zu erhalten, müssten wir also das Blatt mit seinem Bruder verschmelzen. Wenn die Summe der beiden Knoten, die verschmolzen werden, größer als die maximale Anzahl von Einträgen ist, so muss anschließend noch ein erneuter Split durchgeführt werden.
Da jedoch in fast allen praktischen Anwendungen von Datenbank die ’INSERTS’ bei weitem überwiegen, und da die Verschmelzungen und die Splits einen zusätzlichen Aufwand darstellen, lässt man solche Unterschreitungen des Mindestfüllgrades zu und hofft, dass spätere INSERTS dies wieder beheben.

Aufgabe 2: B*-Baum

a)

- Fügen wir eine sortierte Folge von Schlüsseln in einen B*–Baum ein, so erhalten wir ein einziges Blatt mit einer Anzahl von Schlüsseln zwischen m* und 2 _ m*. Alle anderen Blätter enthalten m* viele Schlüssel. Da die maximale Anzahl von Einträgen in den Blättern 2 _ m* ist, erhalten wir einen Füllgrad von ungefähr 50%.

- Für den speziellen Fall des Einfügens von sortierten Folgen existiert in den meisten Systemen die Möglichkeit durch Setzen eines entsprechenden Parameters die spezielle Eigenschaft der Folge für ein effizientes Aufbauen (bzgl. Füllgrad und auch Laufzeit) des B*–Baumes auszunutzen. Dies ist vor allem vorteilhaft, wenn ganze Datenbanken geladen werden. Im laufenden Betrieb kann man den Füllgrad erhöhen, indem man einen Ausgleich zwischen mehreren Geschwisterknoten durchführt. Beim Überlauf eines Knotens wird versucht, Einträge in den Geschwisterknoten einzufügen. So könnte beispielsweise erst ein Split erfolgen, wenn zwei benachbarte Knoten voll sind. Aus den zwei gefüllten Knoten würde dann nur ein neuer Knoten abgesplittet, was in diesem Falle zu einem Mindestfüllgrad von 2/3 führen würde. Im allgemeinen Fall könnte man bei Betrachtung von m–1 um den Einfügeknoten liegenden Geschwisterknoten immer mindestens einen Füllgrad von m/(m+1) erreichen. Da dieses Vorgehen jedoch einen zusätzlichen Aufwand (insbesondere an I/O Aktivität) bewirkt, wird es in der Praxis meistens nicht angewendet.

b)

- Splits können sich im Baum nach oben propagieren, d.h. der Split eines Blattes kann dazu führen, dass der Vaterknoten auch einen Split durchführen muss, usw. Dazu müssen wir den Vaterknoten jedoch kennen. Dies können wir z.B. durch einen solchen Vaterverweis erreichen. Der analoge Fall kann auftreten, wenn wir beim Löschen Knoten zusammenfügen.

- Wird ein innerer Knoten gesplittet, so müssen wir die Vaterverweise aller seiner Kinder ändern und dazu alle entsprechenden Blöcke lesen. Durch den hohen Fanout der Knoten ist dieser Aufwand aber nicht gerechtfertigt. Der hohe Verzweigungsgrad bewirkt außerdem, dass die Höhe des Baumes sehr gering ist (selbst für sehr große Datenbanken kleiner als 6). Daher ist es besser, wenn wir uns den Vater jeweils explizit merken können, oder durch eine rekursive Implementierung der Einfügeoperation dies implizit tun.

c)

Konstante Schlüssellänge

Bei Schlüsseln konstanter Länge kann ein Eintrag direkt über seine Position innerhalb einer Seite adressiert werden. Es sind keine zusätzlichen ’Inhaltsverzeichnisse’ wie TID–Slots nötig. Eine offensichtliche Vorgehensweise ist die lineare Suche, wo die Seite sequentiell von vorne nach hinten durchlaufen wird. Im Mittel sind hier bei m Einträgen m/2 Vergleichsoperationen nötig. Hier braucht keine Sortierung innerhalb der Seite vorzuliegen. Bei einer geordneten Seite kann die Sprungsuche sowie eine binäre Suche durchgeführt werden. Bei der Sprungsuche wird die Seite in n Intervalle fester Länge eingeteilt. Beim ersten Durchlauf wird durch Vergleich mit dem größten Schlüssel jedes Intervalls das Intervall identifiziert, wo der gesuchte Schlüssel liegen muss. Danach wird linear innerhalb des Intervalls gesucht. Bei m Einträgen innerhalb einer Seite sind n/2 + m/(2*n) Vergleiche nötig. Wegen der Optimalität für n _ m _ wird die Sprungsuche oft auch als Quadratwurzel–Suche bezeichnet. Beim binären Suchen sind bei m Einträgen im Mittel log2(m) –1 Vergleiche nötig.

Variable Schlüssellänge

Variable Schlüssellänge kann durch Schlüsseltypen variabler Länge (z.B. VARCHAR(n)) entstehen sowie durch die Kompression von Schlüsselfeldern fester Länge (siehe dazu auch Aufgabe 3). Hier entsteht das Problem, dass die Position eines Eintrags innerhalb einer Seite nicht bekannt ist. Wir können ohne Vorhandensein zusätzlicher Strukturen nur die lineare Suche durchführend. Sind jedoch TID–Slots am Ende der Seite vorhanden, so können wir die Sprungsuche und binäre Suche auf diesem Verzeichnis ausführen. Durch die TID–Slots wird jedoch zusätzlicher Platz innerhalb der Seite belegt, der den Verzweigungsgrad des Baumes reduziert, was den I/O–Aufwand wiederum erhöht. Da I/O–Zugriffe in der Regel wesentlich teurer sind als Vergleichsoperationen, muss hier also sorgfältig abgewogen werden, ob sich die binäre Suche auf Kosten der Speicherreduktion auszahlt.

Aufgabe 3: Präfix B*-Baum

Die Lösung erhält man, wenn man genauso vorgeht, wie es auf den Folien zur Vorlesung beschrieben ist zum Thema *Präfix-B*-Bäume für Strings als Schlüssel*.