HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

B-Baum-Datenstruktur

Bearbeiter 1: Sharif El Deib
Bearbeiter 2: Dominik Petersdorf
Bearbeiter 3: Carlos De Sa e Matos

Gruppe: 48

Ausarbeitung zur Vorlesung Wissenschaftliches Arbeiten

Ort, Abgabedatum

Inhaltsverzeichnis

1

Einleitung

Die Aufgabe von Datenbanken ist die Verwaltung von großen Datenmengen. D.h. man will verschiedene Operationen auf diesen Daten möglichst schnell ausführen können. Man muss die Informationen also in einer Datenstruktur abspeichern, die die entsprechenden Operationen in einer möglichst effizienten Weise unterstützen. Man muss sich also die Frage stellen, welche Datenstruktur hierfür geeignet ist und warum? B-Bäume wurden von Prof. Rudolf Bayer explizit für die Plattenspeicherverwaltung entworfen. Für seine Arbeiten rund um den B-Baum und andere Verdienste erhielt Prof. Rudolf Bayer im Jahr (2001) den „SIGMOD Innovations award“. In unserer Arbeit setzen wir uns unter anderem mit dem Grund auseinander, warum B-Bäume für die Plattenspeicherverwaltung geeignet ist. Dabei ist es zwangsweise notwendig die B-Bäume genauer zu betrachten, und die in dieser Arbeit behandelten Themen näher zu behandeln. Zunächst werden wir die B-Bäume genauer definieren, gefolgt von der Behandlung der Operationen in B-Bäumen. Anschließend werden noch die Komplexitätsklassen angesprochen und eine genauere Beschreibung der Anwendungen.

2

Definition B-Baum

Ein B-Baum ist ein balancierter Suchbaum der Ordnung n, wenn es folgende Eigenschaften enthält:

1. Jeder Knoten besitzt mindestens 1 und höchstens $n - 1$ Elemente.
2. Jeder Knoten mit m Elemente hat $m + 1$ Söhne, außer Blätter.
3. Jeder Wert der Elemente im linken Teil des Baumes ist kleiner als die Werte des rechten Teil des Baumes.
4. Alle Blätter müssen die gleiche Tiefe h haben und dürfen keinen Nachfolger geben.

Nehmen wir einen B-Baum der Ordnung 3 als Beispiel(siehe Abb. 2.1). Jeder Knoten dieses Baumes kann zwischen 1 und 3 Elemente haben. Ein innere Knoten hat im unseren Beispiel 3 Söhne. Die Werten der Elemente erhöhen sich vom linken Teil bis zum rechten Teil des Baumes. Dazu kommt noch, dass jedes Blatt die gleiche Tiefe hat. Im diesen Fall hat jedes Blatt die Tiefe 3. Somit stimmen alle oben genannten Eigenschaften überein.

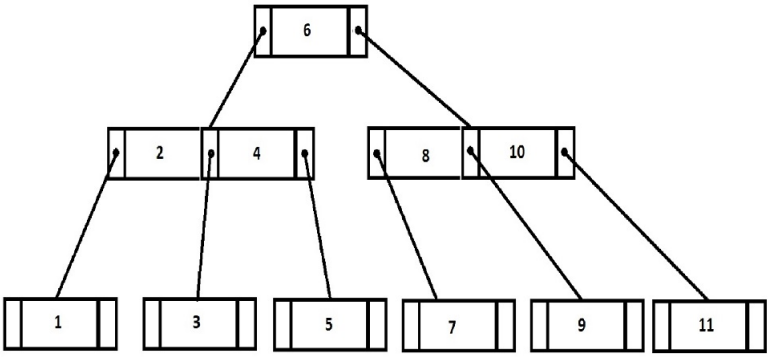

Abb. 2.1. Beispiel eines B-Baumes

Es gibt ein spezieller B-Baum, der 1,2 oder 3 Elemente pro Knoten hat und 2, 3 oder 4 Söhne haben kann. Dieser Baum wird als 2-3-4-Baum gekennzeichnet(siehe Abb. 2.2).

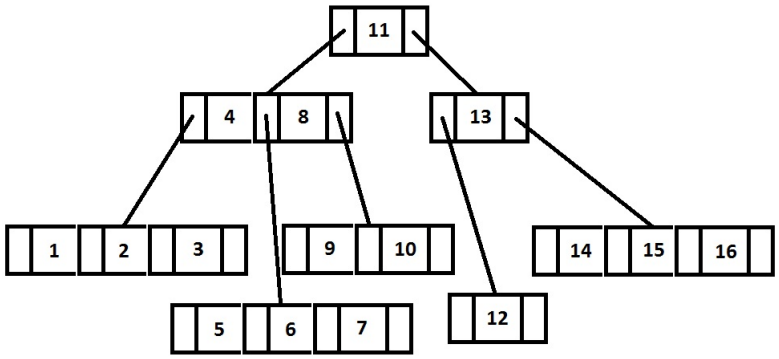

Abb. 2.2. 2-3-4-Baum

3

Operationen

3.1 Suchen

Um einen Schlüssel innerhalb eines B-Baums zu suchen geht man folgendermaßen vor. Ausgehend von der Wurzel werden alle Knoten danach überprüft, ob sich der Schlüssel in dem betrachteten Knoten befindet. Wird er nicht in der Wurzel gefunden wird der kleinste Schlüssel, der größer als der gesuchte Schlüssel ist bestimmt. Sofern dieser Schlüssel existiert, wird bei dem Kindknoten links von diesem weitergesucht. Wenn die Suche am Ende bei einer null-Referenz landet existiert der Schlüssel nicht, oder er wurde nicht gefunden.

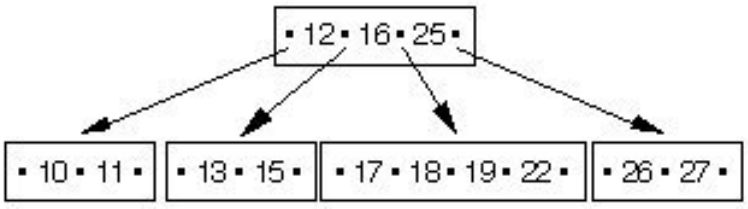

Abb. 3.1. Beispiel B-Baum

3.2 Einfügen

Damit ein neuer Knoten eingefügt werden kann, wird zuerst das Blatt bestimmt in dem sich der Schlüssel befinden müsste. Daher es den Schlüssel noch nicht gibt, bleibt die Suche erfolglos. Nun können 2 Fälle eintreten, wie der neue Schlüssel in den B-Baum eingefügt werden kann. Hierbei ist zu beachten, dass nur $m - 1$ Schlüssel pro Knoten gespeichert werden können.

Fall 1.

Der Knoten k hat m-1 Schlüssel noch nicht gespeichert und es wurde der Schlüssel x auch in der Blattebene nicht gefunden, so wird der Schlüssel in den B-Baum eingefügt. In diesem Fall fügt man den Schlüssel x in dem Knoten k zwischen x_i und x_{i+1} ein.
Ein Beispiel:
Wir fügen in (siehe Abb. 3.1) den Schlüssel 14 ein. Zuerst beginnt der Suchbaum in der Wurzel $-12-16-(12<14>16)$, der Schlüssel wechselt in den Kindknoten zwischen 12 und 16. Dieser Knoten besitzt die Schlüssel $-13-15-$. Daher der B-Baum die Ordnung $m=5$ besitzt, ist in dem Knoten noch Platz für einen weiteren Schlüssel. Dort wird der Schlüssel nach der Definition (siehe Kapitel **??**) eingefügt.(siehe Abb. 3.2)

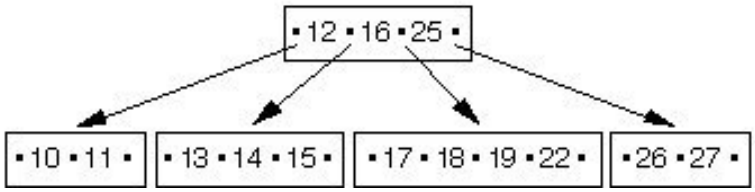

Abb. 3.2. 14 einfügen

Fall 2:

Es wurden bereits $m-1$ Schlüssel in dem Knoten gespeichert, zudem der Schlüssel x gehören würde. Der Schlüssel wird daher zuerst in den betreffenden Knoten seiner Größe nach eingefügt und der zu große Knoten in der Mitte geteilt. Der mittlere Knoten wandert in seinen Vaterknoten und ordnet sich dort seiner Größe nach ein. Die Zeiger auf die Kindknoten passen sich der neuen Anordnung an.
Ein Beispiel:
In die vorherige Abbildung (siehe Abb. 3.2) wird der neue Schlüssel 24 eingefügt. Daher der Knoten zu großwird, teilt er sich und der Schlüssel 19 wandert in den Vaterknoten. Danach bilden sich 2 neue Knoten aus dem zu grossen Knoten.(siehe Abb. 3.3)

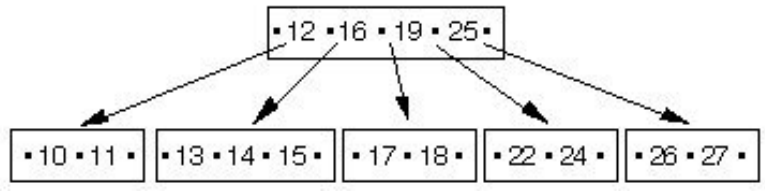

Abb. 3.3. 24 einfügen

3.3 Löschen

Löschen in B-Bäumen ist im Gegenzug zu anderen Bäumen komplexer, da sehr genau zwischen dem Löschen von Knoten und von Blättern unterschieden werden muss. Hierbei kann man 3 Fälle unterscheiden.

1. Wenn das Blatt die Wurzel ist und das Blatt leer ist kann die Wurzel gelöscht werden. Der Baum ist leer.
2. Wenn das Blatt nicht die Wurzel ist, und $m \geq \lceil n/2 \rceil - 1$ Schlüssel besitzt. Dann ist das Löschen beendet.
3. Wenn jedoch $m = \lceil n/2 \rceil - 2$, so muss die Eigenschaft des B-Baums wiederhergestellt werden.
   a) Gibt es einen Nachbarn, der $m > \lceil n/2 \rceil - 1$, so können Schlüssel zwischen den beiden Knoten verschoben werden.
   b) Gilt für beide Knoten jedoch, $m = \lceil n/2 \rceil - 1$, so müssen 2 Knoten vereinigt werden.

Verschiebung von Schlüsseln beim Löschen

Nur rechter Nachbar vorhanden mit $m_{rechts} > \lceil n/2 \rceil - 1$	Linksverschiebung
Linker und rechter Nachbar vorhanden mit $m_{links>} > \lceil n/2 \rceil - 1$ und $m_{rechts>} > \lceil n/2 \rceil - 1$ und $m_{links} m_{rechts}$	Linksverschiebung
Linker und rechter Nachbar vorhanden mit $m_{links>} > \lceil n/2 \rceil - 1$ und $m_{rechts>} > \lceil n/2 \rceil - 1$ und $m_{links} m_{rechts}$	Rechtsverschiebung
nur linker Nachbar vorhanden mit $m_{rechts} > \lceil n/2 \rceil - 1$	Rechtsverschiebung

([Schroeder:03], S. 5)

Vereinigung von Knoten beim Löschen

Vereinigung mit rechtem Nachbarn nach Löschen von a3(siehe Abb. 3.4)

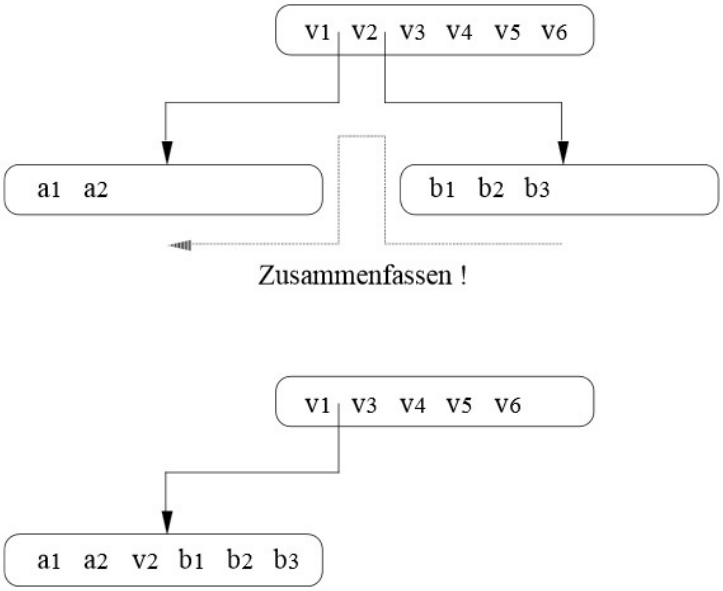

Abb. 3.4. ([Schroeder:03], S. 6)

- Durch das Vereinigen von Knoten wird die Zahl der Werte m im Vorgängerknoten um eins kleiner.
- Daher kann dort wiederum ein Unterschreiten der Untergrenze stattfinden.
- Vereinigungsvorgänge können sich daher bis zur Wurzel fortsetzen.
- ie Wurzel selbst darf jedoch weniger als $\lceil n/2 \rceil - 1$ Werte besitzen, so dass hier der Vorgang endet

4

Komplexität

Wir wollen hier die Zeitkomplexität zum Durchlaufen und Finden eines bestimmten Datenelementes in einem B Baumes untersuchen. Diese gibt an, wovon der Zeitaufwand abhängt, um den B Baum nach einem Datenelement zu durchsuchen. Sowohl die der Zeitaufwand der Operationen Finden, Einfügen und Löschen hängen auch direkt von diesem Zeitaufwand ab. So wie logischerweise zum Löschen eines Datenelementes dieses vor der eigentlichen Operation zuerst gefunden werden muss, muss auch vor dem Einfügen zuerst festgestellt werden an welche Stelle das neue Element eingefügt werden muss, damit die Ordnung der Datenelemente immer noch gewährleistet ist. Diese Ortung der Einfügstelle kommt einem Finden (zum Beispiel des Nachbarelementes) gleich. Bei der exakten Betrachtung von Laufzeiten, müssten viele verschiedene Fälle unterschieden werden, so wird sich oft darauf beschränkt die Komplexitätsklasse des Laufzeitverhaltens zu bestimmen, d. h. es wird bestimmt welcher Klasse von Funktionen die Laufzeitfunktion gehört (linear, quadratisch, logarithmisch...). Die sogenannte Komplexitätsklasse ist im Vergleich zur genauen Laufzeitfunktion einfacher zu bestimmen. Zwar ist eine genaue Berechnung der eigentlichen Laufzeit nicht möglich, aber es ist möglich eine asymptotische Obergrenze der Laufzeit und einen Vergleich zwischen verschiedenen Strukturen und Algorithmen zu erstellen.

4.1 Komplexitätsklasse beim B-Baum

Zunächst müssen wir uns mit der Struktur des B Baumes befassen, um eine Komplexitätsklasse der Operationen auf den B Baum bestimmen zu können. Hierzu zuerst einige Definitionen und Notationen: Man nennt Tiefe t, den Abstand eines beliebigen Knotens zur Wurzel. Man nennt Höhe h eines B Baumes den Abstand von der Wurzel zu den Blättern, es ist also die Tiefe t der Blätter. Die Wurzel ist der Vorfahre jedes Knotens. Ein Blatt ist ein Feld mit Datenelementen und befindet sich am ?Ende? des Baumes, d. H. ein Blatt hat keine Nachkommen (Söhne). Eine spezielle Eigenschaft des B Baumes ist es, dass alle Blätter die gleiche Tiefe T besitzen und stellen somit die Höhe h des B Baumes dar. Um ein Datenelement zu finden, das sich gezwungenerweise in den Blättern befindet, muss ein Algorithmus die Datenstruktur von der Wurzel bis zu dem gesuchten Blatt durchlaufen, also die

Höhe h durchlaufen. Auf dem Weg von einem Knoten zum anderen, muss es einen Speicherzugriff geben, die Anzahl der Speicherzugriffe von der Wurzel zum Blatt ist immer gleich der Höhe h. Folglich ist die Komplexität der meisten Operationen (Find, Insert , Delete) auf B Bäumen proportional zur Höhe h des B Baumes.

Wir wollen nun die Höhe h des B Baumes in Zusammenhang mit der gesamten Anzahl # von Datenelementen im B Baum bringen. Dazu müssen wir entscheiden wie viele Kinder jeder Knoten haben soll, sei 2n diese maximale Anzahl der Kinder. Je nach dem Füllgrad der Knoten, der bei einem B Baum für jeden Knoten zwischen mindestens n und maximal 2n liegt, ergeben sich verschiedene Höhen für die gleiche Anzahl von Datenelementen ?. Die größte Höhe h, d.h. die längste Laufzeit, weist ein B Baum auf, falls jeder Knoten nur die Hälfte der möglichen Kinder aufweist. Dagegen hat ein B Baum mit ? Datenelementen die kürzeste Höhe h, und somit auch die kürzeste Laufzeit, falls jeder Knoten die maximal mögliche Anzahl von Kindern d. h. 2n aufweist. Daher ergeben sich folgende Grenzwerte für h :

$$\log_2 n + 1(\# + 1) - 1 \leq h \leq \log_n + 1(\tfrac{\#+1}{2})$$

5

Anwendung von B-Bäume

Heutzutage werden die B-Bäume in Datenbanksystemen benutzt, die sehr hohe Datenmengen haben. Diese Datenstruktur verkürzt die Zeit die es benötigt, um eine gewählte Menge von Daten zu suchen. Hauptsächlich wird der B-Baum in Luft- und Raumfahrtindustrie eingesetzt.

Ein gutes Beispiel ist ein PC den man zu Hause stehen hat. Ein PC besitzt ein Hauptspeicher und eine Festplatte. Da der Hauptspeicher eine schnellen Lese/Schreiben-Zugriff haben muss, könnte man unmöglich die ganzen Datenmengen in dem Hauptspeicher speichern. Denn dies wurde den Zugriff verlangsamen. Deshalb werden im Hauptspeicher lediglich die Werte der Schlüssel gespeichert und die großen Datenmengen werden auf der Festplatte gespeichert. Die Schlüssel haben die Verweise auf die Sektorblöcke der Festplatte. Dies erhöht die Lese- und Schreibgeschwindigkeit, denn die Zugriff maximiert sich auf die Tiefe des Baumes.

Literaturverzeichnis