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

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. 1.1). 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.

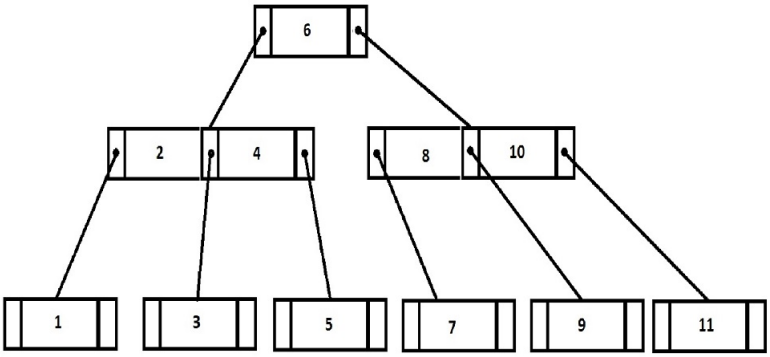

Abb. 1.1. Beispiel eines B-Baumes

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. 1.2).

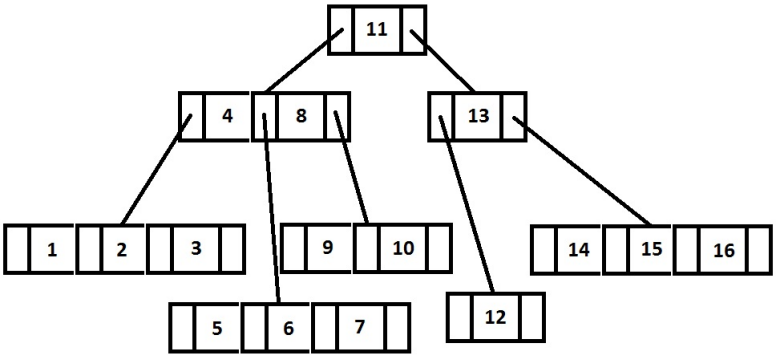

Abb. 1.2. 2-3-4-Baum

2

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 Komplexitätklasse des Laufzeitverhaltens zu bestimmen, d. h. es wird bestimmt welcher Klasse von Funktionen die Laufzeitfunktion gehört (linear, quadratisch, logarhitmisch...). 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.

2.1 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 andern, 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 möglchen 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 :

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

3

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

CDK02. Coulouris, George, Jean Dollimore und Tim Kindberg: *Verteilte Systeme: Konzepte und Design*. Addison-Wesley-Verlag, 2002.

Che85. Cheriton, David R.: *Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems*, 1985.

Mal97. Malte, Peter: *Replikation in Mobil Computing*. Seminar No 31/1997, Institut für Telematik der Universität Karlsruhe, Karsruhe, 1997. `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/1997/31`.

Mos93. Mosberger, David: *Memory Consistency Models*. Technical Report 93/11, University of Arizona, November 1993.

A

Glossar

DisASTer	DisASTer (Distributed Algorithms Simulation Terrain), A platform for the Implementation of Distributed Algorithms
DSM	Distributed Shared Memory
AC	Linearisierbarkeit (atomic consistency)
SC	Sequentielle Konsistenz (sequential consistency)
WC	Schwache Konsistenz (weak consistency)
RC	Freigabekonsistenz (release consistency)