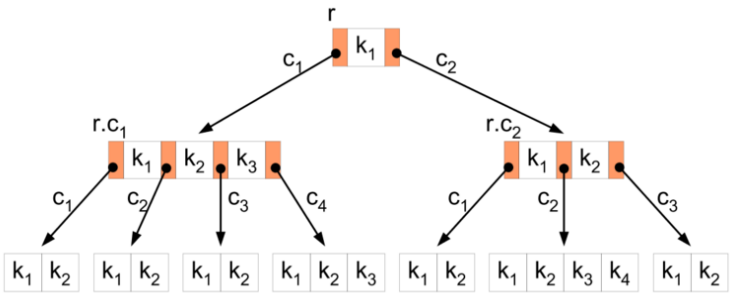
B-Bäume

1 Einleitung

Will man sehr grosse Mengen an Daten verwalten, reichen die einfach Datenstrukturen wie AVL-Bäume oder andere Arten von Binärbäumen nicht mehr aus. Da die Daten aus Speicherplatzmangel in sogenannten Hintergrundspeicher ausgelagert werden muss, benötigt man eine vollständig neue Sichtweise, insbesondere für die Analyse solcher Strukturen.



Strukture eines B-Baumes

Der Speicher wird in *Seiten* organisiert (aufgrund des Aufbaus des Speichermediums). Es können jeweils nur eine konstante Anzahl solcher Seiten im Hauptspeicher gehalten werden, möchte man auf Daten aus einem anderen Block zugreifen, muss diese Seite erst in den RAM geladen werden. Da dieser Vorgang um mehrere Grössenordnungen langsamer ist als ein Zugriff auf normalen Hauptspeicher, will man die Anzahl Zugriffe minimieren.

2 Struktur von B-Bäumen

B-Bäume werden als Vielwegbäume realisiert: Jeder Knoten verfügt über eine bestimmte Anzahl an Schlüsseln und Zeigern, wobei sich diese Werte in bestimmten Schranken bewegen. Damit die Höhe nur logarithmisch wächst, werden an einen **B-Baum der Ordnung m** folgende strukturellen Bedingungen gestellt:

1. Alle Blätter haben die gleiche Tiefe.
2. Jeder Knoten hat mindestens *ceil(m/2)* Söhne. Die Wurzel ist von diese Regel ausgenommen.
3. Die Wurzel hat mindestens 2 Söhne, solange sie kein Blatt ist.
4. Jeder Knoten hat höchstens m Söhne.
5. Jeder Knoten mit *i* Söhnen hat *i-1* Schlüssel.

Diese Bedingungen genügen, um zu garantieren, dass ein Baum höchstens eine in N logarithmische Höhe erreicht.

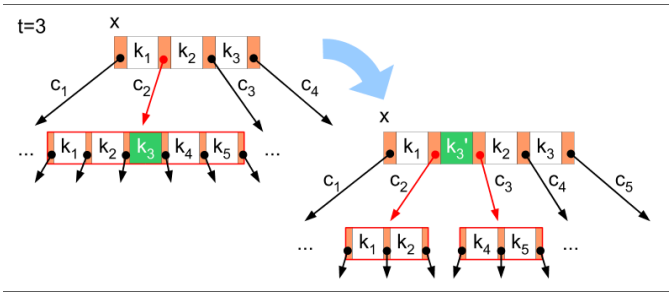
Die Zahl m hängt in der Praxis vom eingesetzten Speichermedium und kann von 100 bis zu mehreren Tausend gehen. Entscheidend ist, wie viel Schlüssel (inklusive Daten) in einer Seite gespeichert werden können, denn typischerweise wird jeweils ein Knoten pro Seite gespeichert.

3 Die elementaren Operationen auf B-Bäumen

3.1 Suchen

Beim Suchalgorithmus in B-Bäumen handelt es sich um die natürliche Erweiterung des Suchens bei Binärbäumen. Man beginnt bei der Wurzel und überprüft dann, ob sich der zu suchende Schlüssel im gerade betrachteten Knoten befindet. Ist dies nicht so und der Knoten ist kein Blatt, so wählen wir den Zeiger zwischen den beiden Schlüsseln, die denen der gesuchte Schlüssel liegt.

3.2 Einfügen



Teilen eines vollen Knotens

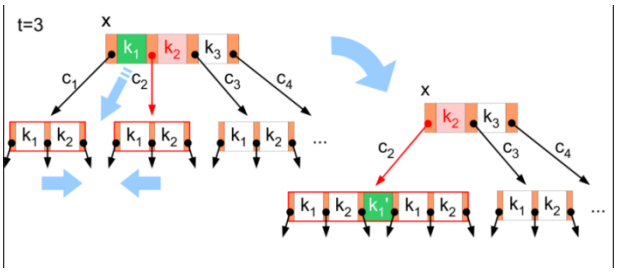
Es wird mit einer (erfolglosen) Suche nach dem einzufügenden Schlüssel x begonnen. Diese endet in einem Blatt, an der zu erwartenden Position für x. Hat der gerade betrachtete Knoten weniger als m-1 Schlüssel, können wir den Schlüssel ganz einfach als neuen Schlüssel in diesen Knoten einfügen. Ist der Knoten jedoch bereits voll, müssen wir ihn teilen. Dazu nehmen wir den Median aller Schlüssel in diesem Knoten und verschieben diesen in den Vaterknoten. Das ergibt eine weitere Unterteilung beim Elternknoten, der jetzt ein Kind mehr hat. Aus dem Kindknoten mit m-1 Schlüsseln wurde nun zwei Knoten mit je m/2 Knoten (Bedingung erfüllt), wobei in einem der beiden nun auch unser Schlüssel x gespeichert werden kann. Das Einfügen eines weiteren Schlüssels im Vaterknoten muss natürlich nicht unbedingt möglich sein, auch diese Knoten kann bereits voll sein. Dann wird einfach ein weiteres Mal geteilt, notfalls bis zur Wurzel. Ist die Wurzel auch voll, erstellt man eine neue, leere Wurzel, die dann nur eine Schlüssel enthält und damit zwei Kinder hat (Bedingung erfüllt).

Ein B-Baum wächst also nicht nach unten, sondern an der Wurzel. Damit ist auch leicht zu sehen, dass die Bedingung (1) erfüllt ist.

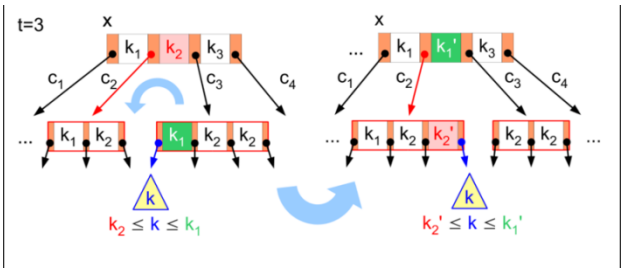
3.3 Löschen

Ähnlich wie beim Einfügen, wo man sich vor zu stark besetzten Knoten schützen musste, kann es hier eine Art Unterlauf geben, sodass ein Knoten weniger als die *ceil(m/2)* Kinder hat. Zudem ist es hier möglich, dass auch innere Knoten bearbeitet werden, was die Sache ein klein wenig komplizierter macht.

Das Vorgehen ist aber nicht grundsätzlich verschieden: Der Schlüssel x wird zuerst gesucht. Handelt es



Verschmelzen zweier Knoten



Verschieben eines Schlüssels

sich dabei um einen Schlüssel eines inneren Knotens, wird der symmetrische Nachfolger von x mit x ausgetauscht, und anstelle dessen der Nachfolger gelöscht. Kann man diesen Schlüssel nun ohne Probleme löschen, sind wir fertig. Entsteht aber ein Unterlauf, so muss etwas angepasst werden. Als erstes werden die beiden Brüder des Knotens mit dem Schlüssel x, nennen wir ihn n, angesehen. Hat einer von diesen beiden mehr als die minimale Anzahl Schlüssel, können Schlüssel von diesem Knoten transferiert werden (siehe Graphik). Ist dies bei beiden nicht der Fall, müssen n und irgendein Nachbar verschmolzen werden. Das geht, da zuvor beide *ceil(m/2)-1* Schlüssel hatten, und nun n einen Schlüssel verlor. Damit hat der entstehende Knoten nach der Verschmelzung höchstens m-1 Schlüssel.

4 Varianten der B-Bäume

4.1 B+-Bäume

Es ist natürlich auch möglich, die Daten nur in den Blättern des Baumes zu speicher, also den B-Baum als Blattsuchbaum zu realisieren. Dabei wird dann immer ein zusätzlich Blockzugriff nötig, um an die Daten zu kommen, dafür können pro „page“ im Speicher viel mehr Schlüssel gespeichert werden, da die Daten nicht bei den Schlüsseln gespeichert werden müssen. Damit wird der „branching factor“ maximiert.

4.2 B*-Bäume

Es handelt sich hierbei um eine weitere Erweiterung von B-Bäumen. Wie in B+-Bäumen werden die Daten nur in den Blättern gespeichert, doch hier gibt es zusätzlich noch die Bedingung, dass jeder Knoten zu mindestens 2/3 voll sein muss.