Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(19 - B-Bäume)

Prof. Th. Ottmann

B-Bäume

Motivation:

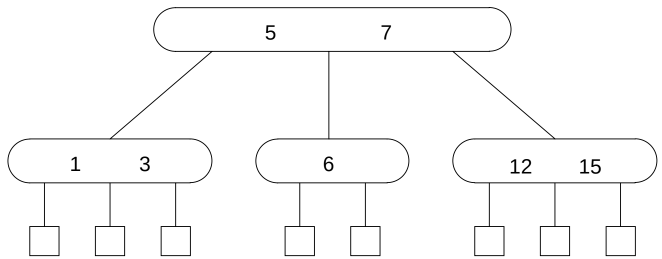
- Bisher waren wir davon ausgegangen, dass die durch Bäume strukturierten Daten im Hauptspeicher Platz finden.
- In der Praxis (z.B. in Datenbanken) ist man jedoch häufig gezwungen, die Daten und auch die Schlüssel auf externen Medien (heute in der Regel Festplatten) abzulegen.
- Ziel von B-Bäumen ist, die Anzahl der Zugriffe auf das externe Medium zu reduzieren.

Definition der B-Bäume

Definition: Ein Baum heißt B-Baum der Ordnung *m*, wenn die folgenden Eigenschaften erfüllt sind.

1. Jeder Knoten mit Ausnahme der Wurzel enthält mindestens $\lceil m / 2 \rceil$ -1 Daten. Jeder Knoten enthält höchstens m - 1 Daten. Die Daten sind sortiert.
2. Knoten mit k Daten $x_1, \dots, x_k$ haben $k + 1$ Referenzen auf Teilbäume mit Schlüsseln aus den Mengen $\{-\infty, \dots, x_1 - 1\}, \{x_1 + 1, \dots, x_2 - 1\}, \dots, \{x_{k-1} + 1, \dots, x_k - 1\}, \{x_k + 1, \dots, \infty\}$.
3. Die Referenzen, die einen Knoten verlassen, sind entweder alle null-Referenzen oder alle Referenzen auf Knoten.
4. Alle Blätter haben gleiche Tiefe.

Ein Beispiel



Hinweis: Im Kontext von B-Bäumen sind null-Referenzen durch □ dargestellt.

Beobachtung: Dieser Baum hat 7 Schlüssel und 8 null-Referenzen.

Anzahl der null-Referenzen

Behauptung: Die Anzahl der null-Referenzen ist stets um eins größer als die Anzahl der Schlüssel in einem B-Baum.

Beweis:

Induktionsverankerung: Der leere B-Baum enthält eine null-Referenz aber keinen Knoten.

Induktionsvoraussetzung: Die Aussage gelte für alle B-Bäume mit höchstens *n* - 1 Schlüsseln.

Induktionsschluss: Sei p die Wurzel eines B-Baums mit *n* Schlüsseln. Angenommen *p* hat *k* - 1 Schlüssel und *k* Teilbäume mit $l_1, \ldots, l_k$ Schlüsseln. Nach Induktionsvoraussetzung ist die Summe der null-Referenzen in den Teilbäumen genau

$$\sum_{i=1}^{k} (l_i + 1) = 1 + (k - 1) + \sum_{i=1}^{k} l_i = 1 + n$$

Höhe eines B-Baums

- Um die Anzahl der in einem B-Baum mit Höhe *h* gespeicherten Schlüssel abzuschätzen, genügt es also, die Anzahl der null-Referenzen zu bestimmen.
- Offensichtlich ist die Höhe maximal, wenn in der Wurzel lediglich ein und in jedem weiteren Knoten genau $\lceil m / 2 \rceil$- 1 Schlüssel enthalten sind.
- Die minimale Anzahl von null-Referenzen ist somit

$$N_{\min} = 2 * \left\lceil \frac{m}{2} \right\rceil^{h-1}$$

- Ist ein B-Baum mit *N* Schlüsseln und Höhe *h* gegeben, so hat er (*N* + 1) null-Referenzen und es muss gelten:

$$N_{\min} = 2 * \left\lceil \frac{m}{2} \right\rceil^{h-1} \leq (N + 1) \Rightarrow h \leq 1 + \log_{\left\lceil \frac{m}{2} \right\rceil} \left(\frac{N + 1}{2} \right)$$

Konsequenzen

- B-Bäume haben demnach auch die für balancierte Bäume typische Eigenschaft, dass die Höhe logarithmisch beschränkt ist in der Anzahl der gespeicherten Schlüssel.
- Für *m* = 199 haben B-Bäume mit 1.999.999 Schlüsseln höchstens die Höhe 4.

Suchen in einem B-Baum

Um einen Schlüssel in einem B-Baum zu suchen, verfahren wir wie folgt:

1. Ausgehend von der Wurzel prüfen wir, ob der gesuchte Schlüssel sich in dem gerade betrachteten Knoten befindet.
2. Ist das nicht der Fall, bestimmen wir den kleinsten Schlüssel der größer als der gesuchte ist.

   Existiert dieser Schlüssel, gehen wir zum Nachfolger-Knoten links von diesem Schlüssel über.

   Existiert der Schlüssel nicht, gehen wir zum letzten Nachfolger-Knoten über.
3. Wenn wir bei einer *null*-Referenz landen, ist die Suche erfolglos.

Suche innerhalb eines Knotens

Hier sind prinzipiell verschiedene Verfahren denkbar:

- lineare Suche
- binäre Suche
- . . .

Allerdings beeinträchtigt die Suche innerhalb eines Knotens die Rechenzeit eher wenig, weil diese hauptsächlich durch die Anzahl der Zugriffe auf den Hintergrundspeicher (Festplatte) beeinflusst wird.

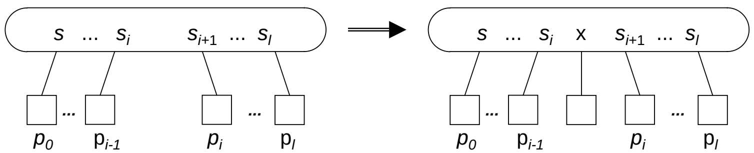
Einfügen eines Schlüssels in den B-Baum

- Zunächst suchen wir die Stelle, an der der Schlüssel x im Baum vorkommen sollte.
- Ist die Suche erfolglos, endet sie in einem Blatt p an der erwarteten Position von x.
- Seien s_1, ... , s_l die in *p* gespeicherten Schlüssel.
- Wir unterscheiden 2 Fälle:

1. Das Blatt ist noch nicht voll, d.h. $l < m - 1$.
2. Das Blatt ist voll, d.h. $l = m - 1$.

Fall 1: Das Blatt ist noch nicht voll

- Wir fügen *x* in dem Blatt *p* zwischen dem Schlüsselpaar $(s_i; s_{i+1})$ ein, für das $s_i < x < s_{i+1}$. Ist *x* kleiner als s_0 oder größer als s_l, so fügen wir *x* am Anfang oder am Ende ein.
- Dann setzen wir den Wert der ebenfalls einzufügenden Nachfolgereferenz auf *null*.

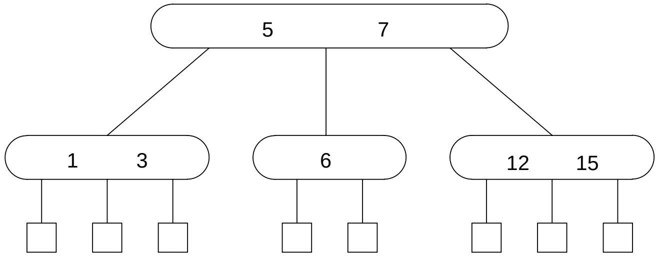


Fall 2: Das Blatt ist voll und enthält genau *m* -1 Schlüssel

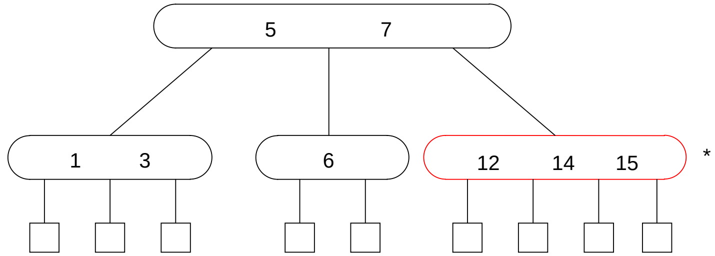
- Wir ordnen *x* in die Folge der Schlüssel entsprechend der Ordnung ein.
- Seien $k_1, \dots, k_m$ die Schlüssel in *p* in aufsteigender Reihenfolge in *p*.
- Wir erzeugen nun ein neues Blatt *q* und verteilen die Schlüssel so auf *p* und *q*, dass *p* die Schlüssel $k_1, \dots, k_{\lceil m/2 \rceil - 1}$ und *q* die Schlüssel $k_{\lceil m/2 \rceil + 1}, \dots, k_m$ enthält.
- Der Schlüssel $k_{\lceil m/2 \rceil}$ wandert in den Vorgänger φ*p* von *p*.
- Wenn *j* die Position von $k_{\lceil m/2 \rceil}$ in φ*p* ist, wird p_{j-i} eine Referenz auf *p* und p_j eine Referenz auf *q*.
- Dieses Verfahren wird rekursiv fortgesetzt, bis wir an der Wurzel angelangt sind und ggf. eine neue Wurzel mit einem Schlüssel erzeugt haben.

Ein Beispiel (2-3-Baum)

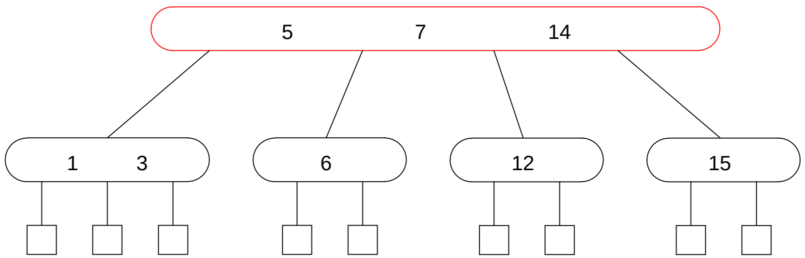
Ausgangssituation:



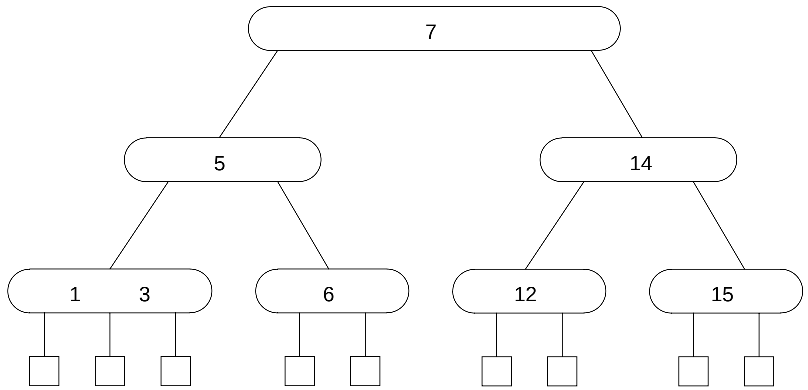
Einfügen von 14



Aufteilen des Blattes



Aufteilen der Wurzel



Löschen von Schlüsseln aus einem B-Baum

- Das Löschen von Schlüsseln ist leider wieder komplizierter als das Einfügen.
- Beim Entfernen unterscheiden wir, ob der zu löschende Schlüssel *x* aus einem Blatt oder aus einem inneren Knoten gelöscht werden soll.
- Offensichtlich besteht das Problem, dass beim Löschen eines Schlüssels aus einem Knoten ein Underflow auftreten kann, d.h. dass in dem Knoten zu wenig Schlüssel vorhanden sind.

Löschen aus einem Blatt

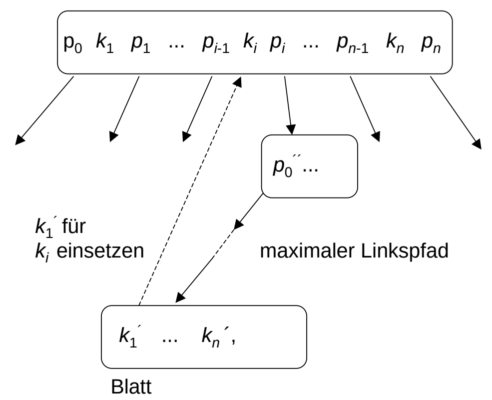
- Ein Blatt hat die Struktur

$$null,\ k_1,\ .\ .\ .\ ,\ null,\ k_i,\ null,\ .\ .\ .\ ,\ k_l,\ null$$

- Wenn nun das Element mit Schlüssel k_i entfernt werden soll, so streicht man einfach k_i und die darauf folgende null-Referenz.
- Ein Underflow tritt auf, falls $l = \lceil m / 2 \rceil - 1$ war.

Entfernen aus einem inneren Knoten

- Im Unterschied zum Löschen aus einem Blatt haben alle Referenzen einen Wert ungleich *null*.
- Insbesondere existiert ein Nachfolger p_i rechts von dem zu löschenden Schlüssel k_i.
- Demnach ist der symmetrische (oder inorder) Nachfolger von k_i im durch p_i referenzierten Teilbaum zu finden (analog zu binären Suchbäumen).
- Wegen der Ausgeglichenheit muss der symmetrische Nachfolger von k_i in einem Blatt sein.
- Wir ersetzen nun k_i durch seinen symmetrischen Nachfolger und löschen k_i aus dem Blatt (erneut Fall 1).



q

φp

Behandlung des Underflows

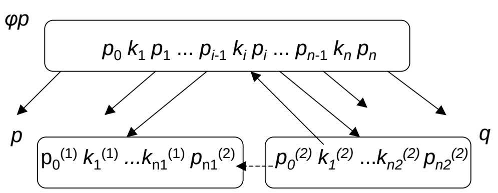
- Bei der Behandlung des Underflows sind wieder verschiedene Fälle möglich.
- Wir benötigen zwei Operationen:

  1. das Ausgleichen (zwischen zwei Bruder-Knoten)
  2. das Verschmelzen (von zwei Bruder-Knoten)

Ausgleich zwischen Bruder-Knoten

- Zwischen zwei Brüdern p und q wird ausgeglichen, wenn bei p ein Underflow eintritt und q mehr als $\lceil m / 2 \rceil - 1$ Schlüssel enthält.
- Idee der Austauschoperation ist, aus *q* einen Schlüssel zu entfernen und diesen in den Vorgänger φ*p* (von *p* und *q*) zu einzufügen. Aus φ*p* entnehmen wir dann einen Schlüssel, den wir in *p* einfügen.
- Dies erfolgt so, dass die B-Baum-Eigenschaft erhalten bleibt.
- Da das Problem symmetrisch ist, nehmen wir an, dass *p* linker Bruder von *q* ist.

Die Ausgleichsoperation

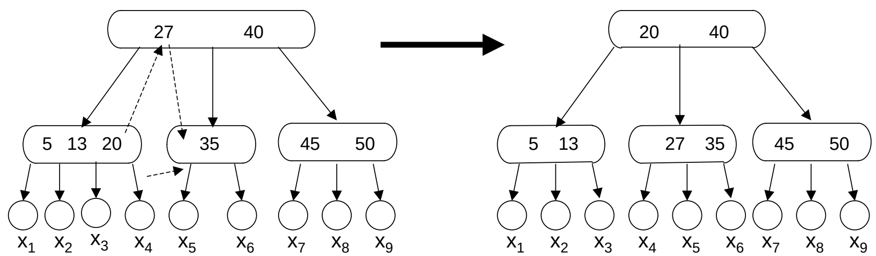


1. k_i wird größter Schlüssel in Knoten *p*.
2. $p_0^{(2)}$ wird letzte Referenz in Knoten *p*.
3. k_i wird in φ*p* durch $k_1^{(2)}$ aus *q* ersetzt.

Hinweis: Der Baum hat anschließend wieder die B-Baum-Eigenschaft.

Ein Beispiel

3-6-Baum mit Underflow im zweiten Nachfolger der Wurzel:



Das Verschmelzen von Bruder-Knoten

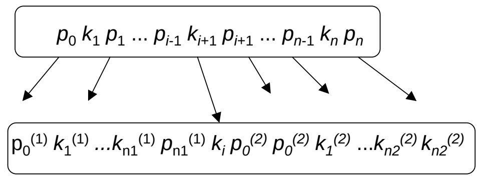
- Ein Knoten *p* wird mit einem seiner Brüder verschmolzen, wenn bei *p* ein Underflow eintritt und kein direkter Bruder von *p* mehr als $\lceil m/2 \rceil - 1$ Schlüssel enthält.
- Idee der Verschmelzungsoperation ist, aus *p* und seinem Bruder *q* einen neuen Knoten erzeugen. Zwischen die Schlüssel von *p* und *q* kommt jedoch der entsprechende Schlüssel aus dem Vorgängerknoten φ*p*.
- Danach hat der neue Knoten

$$\lceil m/2 \rceil - 2 + \lceil m/2 \rceil - 1 + 1 = 2 * \lceil m/2 \rceil - 2 \leq m - 1$$

Knoten.

- Allerdings haben wir aus φ*p* einen Knoten entnommen, so dass dort ggf. ein Underflow eintreten kann.
- Da das Problem symmetrisch ist, nehmen wir erneut an, dass *p* linker Bruder von *q* ist.

Die Verschmelzungsoperation



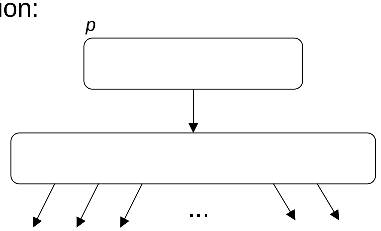
mit $n_1 = \lceil m/2 \rceil - 2$ und $n_2 = \lceil m/2 \rceil - 1$

Das Vorgehen:

- k_i aus φ*p* wandert in den neuen Knoten.
- Der neue Knoten wird Nachfolger p_i -1 in φ*p*.
- p_i wird aus φ*p* gelöscht.

Spezialfall Wurzel

- Die Verschmelzung wird rekursiv nach oben fortgesetzt, bis wir ggf. bei der Wurzel ankommen.
- Wenn wir dann aus der Wurzel den letzten Schlüssel entfernen, haben wir folgende Situation:



- Da die Wurzel keinen Schlüssel mehr enthält, können wir sie einfach löschen und ihren einzigen Nachfolger als Wurzel verwenden.
- Die Höhe des B-Baums ist dann um 1 gesunken.

Ein Beispiel für eine Verschmelzung

3-6-Baum mit Underflow im zweiten Nachfolger der Wurzel: